Hochschule
Ravensburg-Weingarten
University of Applied Sciences

Development of a control box for a mechanical pressure scanner

Winter semester 2013-14

Student: Andrea Pérez López

Supervisor: Prof. Dr. –Ing. Gerd Thieleke

Acknowledgements

I would like to express my gratitude to my supervisor Prof. Dr.-Ing. Gerd Thieleke for his oversight and for let me enjoy this great experience.
I would like to thank Professor Ludescher for taking time out from his busy schedule to help us to solve our problems always that we have needed him.
The programming part would have been much harder without the great help of my good friend Dodel Hidayatodin. Thanks a lot.
I must also acknowledge my colleague Moisés González for his contribution and time dedicated with me in the laboratory.
Last but not the least, I would like to thank my family, my friends and my boyfriend for the support they provided me through this wonderful challenge.
In conclusion, I recognize that this research would not have been possible without each and every one of them. My sincerest thanks.

Contents

INTRODUCTION *1*

HARDWARE *3*

Metallic board 3

Mechanical multiplexer 4

Inductive pressure transducer 7

Supply system 8

Optocoupler 9

Box description 10

Amplifier 13

Filter 14

Microcontroller 15

LCD display 18

SOFTWARE *19*

AVR Atmega 16 19

LabView 22

Introduction 22

Program 22

CONCLUSION *23*

REFERENCES *24*

ANNEXES *25*

Annex A: manufacturing of the printed circuit of the filter 25

Annex B: circuits diagram and layout of the boards manufactured by us 28

Amplifier 28

Filter 29

Optocoupler 30

Annex C: microcontroller programming 31

Main program 31

LCD.c subprogram 32

LCD.h subprogram 39

Uart.c subprogram 42

Uart.h subprogram 48

ADC.c subprogram..... 50
ADC.h subprogram 51
Multiplexer.c program 51
Multiplexer.h program 53

List of figures

Figure 1: block diagram 2
Figure 2: prototype design 3
Figure 3: mechanical multiplexor 4
Figure 4: position transmitter 5
Figure 5: solenoid stepper drive 6
Figure 6: mux support 6
Figure 7: inductive pressure transducer 7
Figure 8: dual-channel optocoupler scheme 9
Figure 9: inside the box 11
Figure 10: box 11
Figure 11: theoretical amplifier diagram 13
Figure 12: theoretical filter diagram 14
Figure 13: ATmega16 Pin Diagram 17
Figure 14: LCD pin configuration 18
Figure 15: printed circuit diagram in a glossy paper Figure 16: ironing the filter board 25
Figure 17: after ironing the filter board Figure 18: board submerged in ferric chloride 26
Figure 19: unnecessary copper removed 26
Figure 20: toner removed Figure 21: drilling 27
Figure 22: finished filter Figure 23: drilled board 27
Figure 24: amplifier circuit diagram 28
Figure 25: filter circuit diagram 29
Figure 26: filter layout 29
Figure 27: optocoupler circuit diagram 30
Figure 28: optocoupler layout 30

INTRODUCTION

In broad terms, the objective of this thesis is to be able to measure and analyse pressure signals coming from different points of a turbine. For this purpose it is needed to read and control a signal, that travels from the turbine to a mechanical multiplexer, which can receive a maximum of 48 pressure inputs and even though it has only one output, it is able to rotate and vary the output pressure which enables the analysis of different pressures in 48 alternate locations of our analysed turbine. This pressure signal, coming from the multiplexor, is sent to the pressure scan which transforms the pressure from a mechanical input into an analogue signal between -10 V and 10V. The signal is then adapted by an amplifier and a filter so that the microcontroller can receive and read it. Then the microcontroller generates a digital signal that will be afterwards read by an external PC via RS232 serial port using LabView to analyse the pressure signals.

On the other hand, the multiplexor can rotate by two different means. The first one is just by pushing manually two switches called “manual step” (increase one position) and “manual home” (come back to 48, also called 0 or Home position) located at the optocoupler board. The other way is by sending the order, of step and home, from LabView to the microcontroller through the RS232 interface.

Apart from this, we were also committed to find out the functionality of some cables which are coming from the position transmitter (belongs to the multiplexor) that are currently unconnected and study how to know exactly when the multiplexor is in home position.

Below this paragraph, it is shown a block diagram of the whole system to provide a generic view to readers.

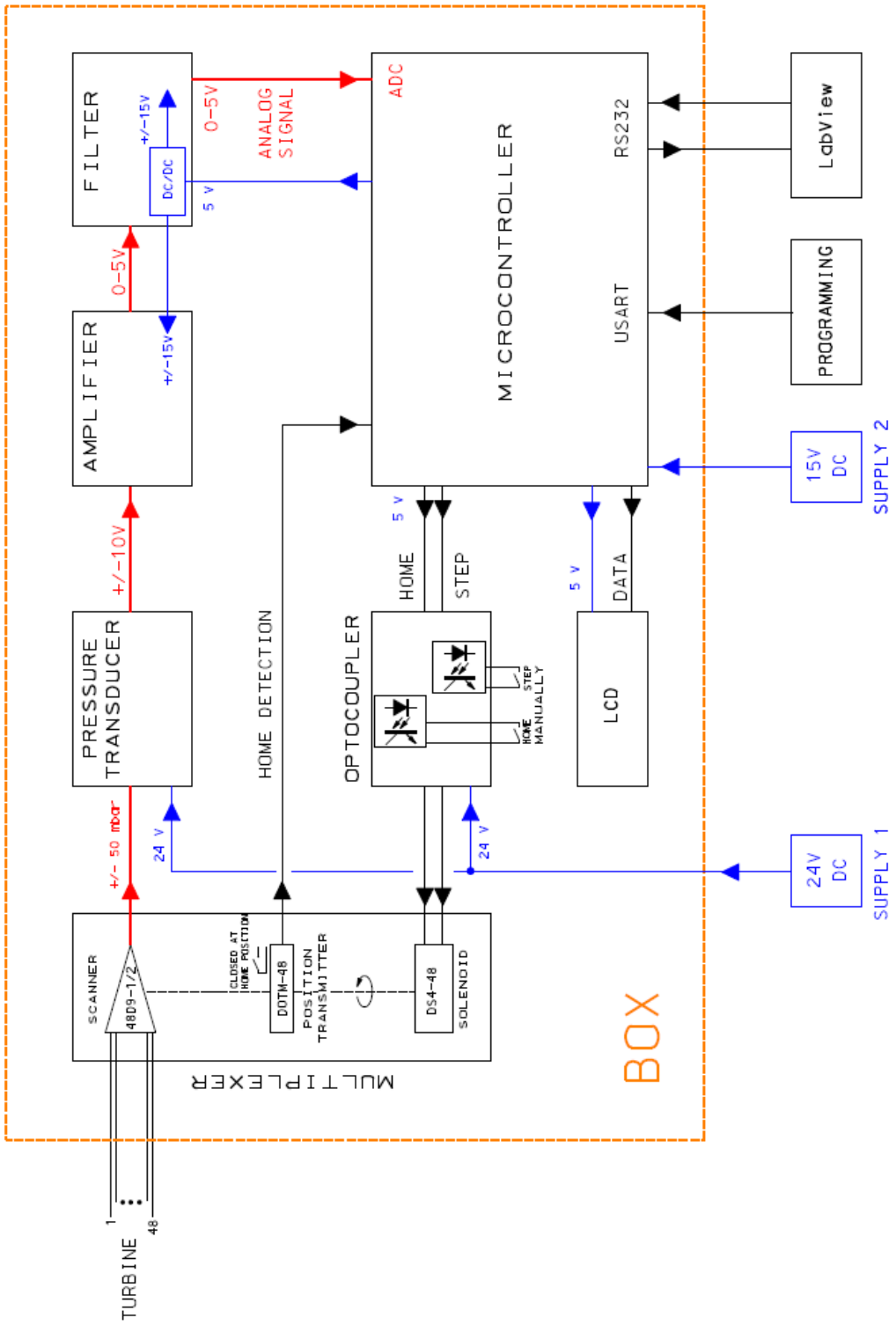


Figure 1: block diagram

HARDWARE

Metallic board

Below this paragraph, it is shown a squeme that I made related to the distribution of some different devices needed for this task which are: Mechanical multiplexer, Optocoupler, Inductive pressure transducer and Supply system. I decided to locate two plastic boxes with the same dimensions, 80x100 (all the measures are in millimeters), in order to provide more flexibility in case that a change in the layout is required.

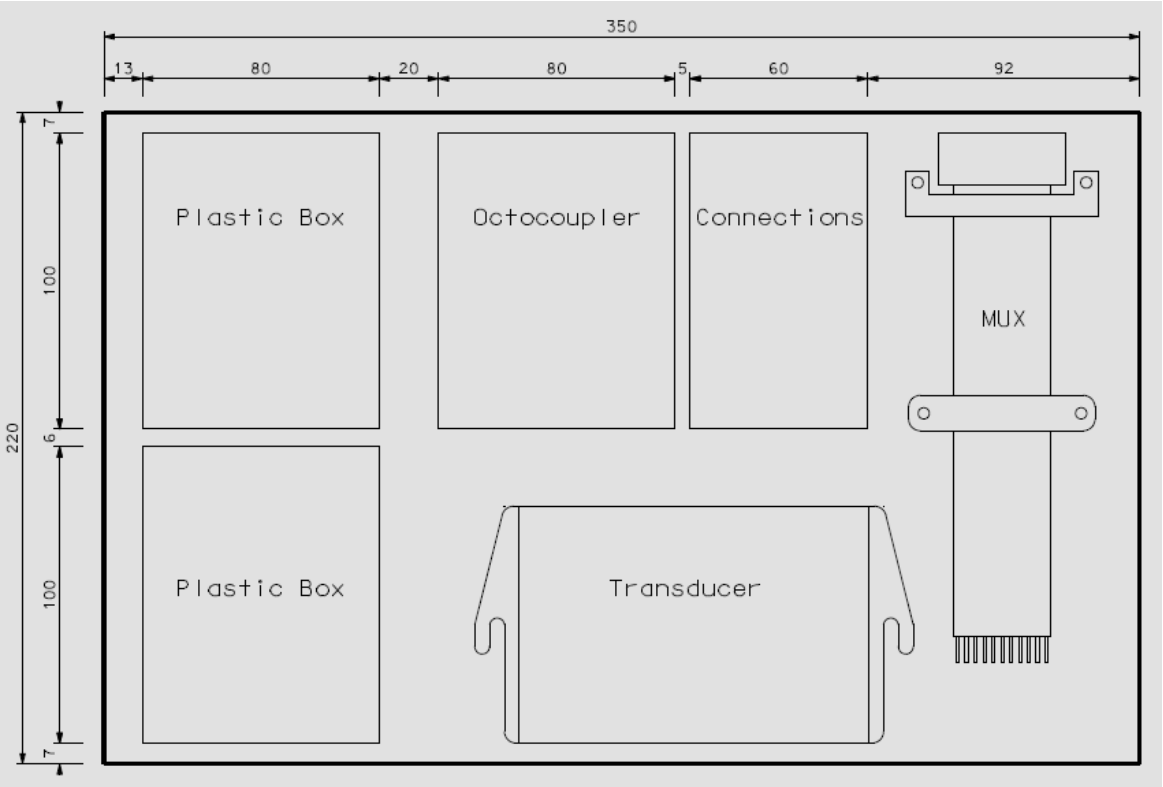


Figure 2: prototype design

Mechanical multiplexer

A multiplexer (or mux) is a device which allows to read and transmit several input signals sequentially thanks to a solenoid that allows to rotate the mux. In my case the multiplexer was already done and was chosen. It is the electro-mechanical pressure multiplexer “DS4-48” of the company Scanivalve which has 48 pressure inputs and one pressure output. The main advantage of it is that it lets us measure sequentially the pressure in 48 different points in a turbine only using this multiplexor.

The pressure scanner is controlled by a microcontroller and can be operated using LabView or manually using switches (“step” and “home”). If step button is pushed, 24V arrives to the line of "step" so the mux rotates and changes to the next channel and the pressure output will be the pressure found in this new point. If it arrives at channel 47, it is followed again by channel 48 (or 0). In order to return to position number 0, it is only needed to push “home” button and the line "home" will be connected to 24V making the multiplexer switch through all the positions until it reaches the first channel (position zero).

This multiplexor consists of three parts:

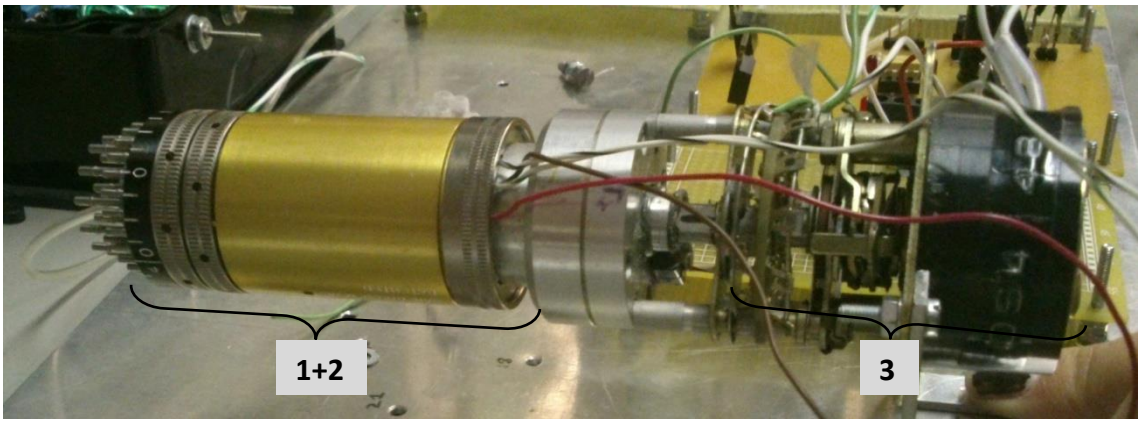


Figure 3: mechanical multiplexor

1. **Scanner** with connector mod. 48D9-1/2 for the pressures application that has a range from 0,1 up to 100 psi, with up 48 lines of vinyl tubing.
2. Position transmitter odd-even mod. DOTM-48. It is also called **odd-even tube marker (OETM)**. This transmitter generates square waves which

alternate from odd numbered parts to even parts. The odd square waves are brought out on a separate wiper from the even square waves.

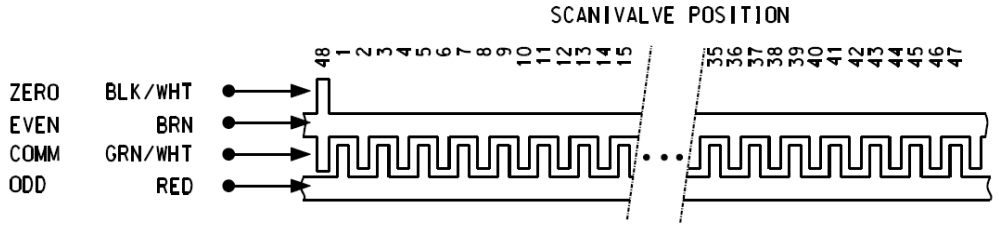


Figure 4: position transmitter

For this project, the four cables didn't have to be used. Nonetheless, I tested them to make sure they weren't damaged and it was a success:

- First of all, I pushed "home" and then, with a multimeter, also called VOM (Volt-Ohm meter), I measured the continuity between comm. and zero, comm. and even and the open circuit between comm. and odd (because home is an even number and is zero). For this purpose, I fixed one point in comm. and the other point of the millimeter was changing depending on which cable we wanted to check.
- Right after, we pushed step once, so the multiplexor was in position 1. In this case we measured the continuity in comm.-oven and short circuit in comm.-even and comm.-zero. Indeed, it was in this way in all even numbers.
- Finally, odd numbers were checked successfully. As you can assume, in this case I measured the continuity comm.-odd and short circuit in comm.-even and comm.-zero.

As I said in the introduction, the verification of home position was needed. For that, I used zero and comm. cables which were used as a feedback to ensure that the mux was at home. It will be explained after in the microcontroller section of this report.

3. **Solenoid** stepper drive DS4-48.

   To advance or retard the entire transmitter, loosen the hub set screw ¼ turn and turn the hub relative to the Scanivalve drive shaft the proper amount. When the transmitter is correctly phased, the line scribed in the tub face should be approximately parallel with the drive shaft slot.

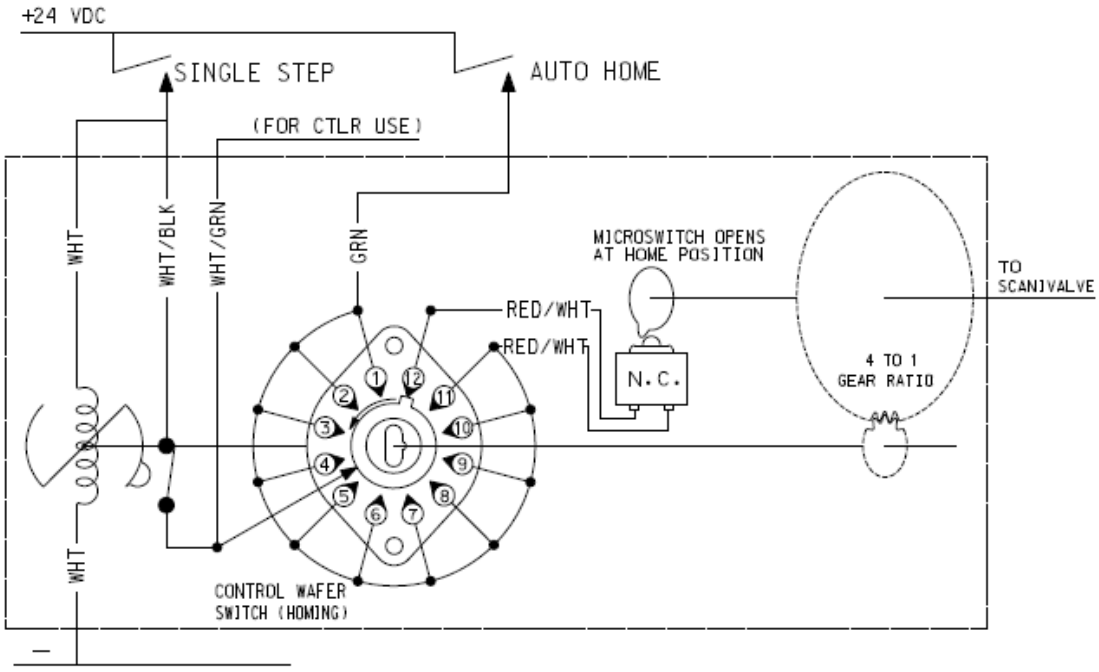


Figure 5: solenoid stepper drive

The multiplexor was not totally fixed and there were little movements due to vibrations. To avoid that, as improvement, I decided to manufacture a piece which could absorb these vibrations and avoid all kind of movements, vertically as well as horizontally. For that, I was inclined to make a circular shape surrounding the position transmitter. It could be adjusted by two crews. The chosen materials are steel for providing rigidity and an elastic material recovering it for buffering vibrations. In the next picture you can see it in detail:

Figure 6: mux support

Inductive pressure transducer

The main function of a pressure transducer is to transform a pressure signal input to an analog voltage output. In this case the pressure transducer already selected was the device “Niederbereich-Druckaufnehmer Typ DPS” of the company Althen. This device allows transmitting a pressure input range of +/- 50 mbar by means of membranes that are sensitive to that pressure range. This pressure signal is performed in a voltage output signal, which varies linearly with pressure changes over the operating range, from -10 V up to 10 volts. The mechanism used for the transmission is an inductive system which performs a contactless scan of those membranes and the output signal is generated.

Therefore, the inductive pressure transducer receives the pressure signal from the mechanical multiplexer and transforms this signal into an electrical signal that goes to the amplifier.

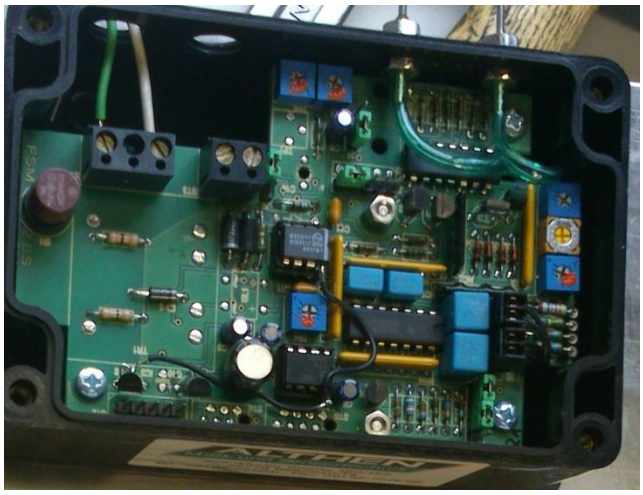


Figure 7: inductive pressure transducer

Supply system

For this project, a supply board was already designed with a piece which is difficult to get. Hence, the supply system is redesigned with the supervisor's approval.

In electronics, a power supply is a device that converts AC voltage in one or more continuous voltages feeding the electronic devices which it connects to. I got two AC/DC devices: one feeds the metallic board out the box and the other one feeds the devices in the box in order to separate the two grounds and to isolate low voltage system from the high voltage one.

The AC/DC device that I got for the box transforms 220 AC volts into 15 DC volts. This 15V arrives to the microcontroller board where there are transformed in 5 V to enable to run our microcontroller. These pins with 5 volts feeds a DC/DC convertor located in the filter board whose input is 5 V and output -/+ 15 V. With these -/+ 15 V the filter and the operational amplifier boards are powered.

The AC/DC device used for feeding the metallic board transforms 220 AC volts into 24 DC volts. It supplies the optocoupler, multiplexor and pressure transducer.

If a clarification is needed, please return to *Figure 1: block diagram*

Optocoupler

In broad terms, an optocoupler, also called opto-isolator, photocoupler or optical isolator, is a component whose function is to transfer electrical signals by means of light. Therefore, they keep two circuits electrically isolated to prevent high voltages from affecting the system receiving the signal. A common type of optocoupler consists of a LED and a phototransistor in the same package, like the used in our project. The kind of optocouplers exactly used have been ILD 620 dual channel, it means that in every optocoupler device there are two independent optocoupler circuits as can be seen in the next scheme:

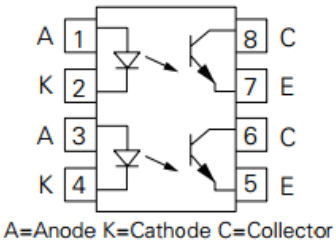


Figure 8: dual-channel optocoupler scheme

These multi-channel phototransistor optocouplers use GaAs IRLED emitters and high gain NPN silicon phototransistors. Some of the advantages of using multi-channel optocouplers are:

- Reduced board space
- Lower pin and parts count
- Better channel to channel CTR match
- Improved common mode rejection

I had to manufacture a board (see Annex A: manufacturing of the printed circuit of the filter) where two dual-channel optocouplers are located, one will be used for step and home signals (explained below) and the other one initially was thought for the cables zero and comm. but after manufacturing the board I discovered that the cables comm. and zero were isolated, so that it was not needed to connect them to the optocoupler to separate grounds. Nonetheless, as the board was already done, I decided to keep it for future improvements or extensions of the project.

As it has been said, only one dual-channel optocoupler has been used for step and home signals (see *Figure 27: optocoupler circuit diagram*), which are sent from the

microcontroller and they arrive to the multiplexer through the optocoupler varying its position. These two optocouplers are used to separate the 24 volts and 3 amperes that the multiplexer uses from the microcontroller which works with 5 volts.

Apart from the optocoupler, in this board there are two data lines receiving information from the microcontroller for rotating the mux. Besides there are four pairs of pins that can be short circuited to operate the multiplexer manually. Two of them are used: one pair is for step function and the other one for home. There are also four led diodes that indicate if a switching signal from the input circuit (from the microcontroller) is present and another four led diodes are used to indicate if the NPN MOSFET transistors have switched. It was needed to provide heat sinks for the transistors due to the high temperatures that they were working.

The pressure transducer is also connected to this board to be powered with 24 volts.

Box description

We were provided with a box to transport the filter, amplifier and microcontroller boards easily. The front panel was already designed. We kept it, even though we only use two of all the holes done, just in case they are needed in future. For esthetic reasons, we decided to recover the whole frontal panel with a grey sticker leaving only visible the place where the LCD display and rs232 serial port will be located.

Below these lines, there is an image showing the metallic board which will be inside the box and the box itself with the beautified frontal panel:

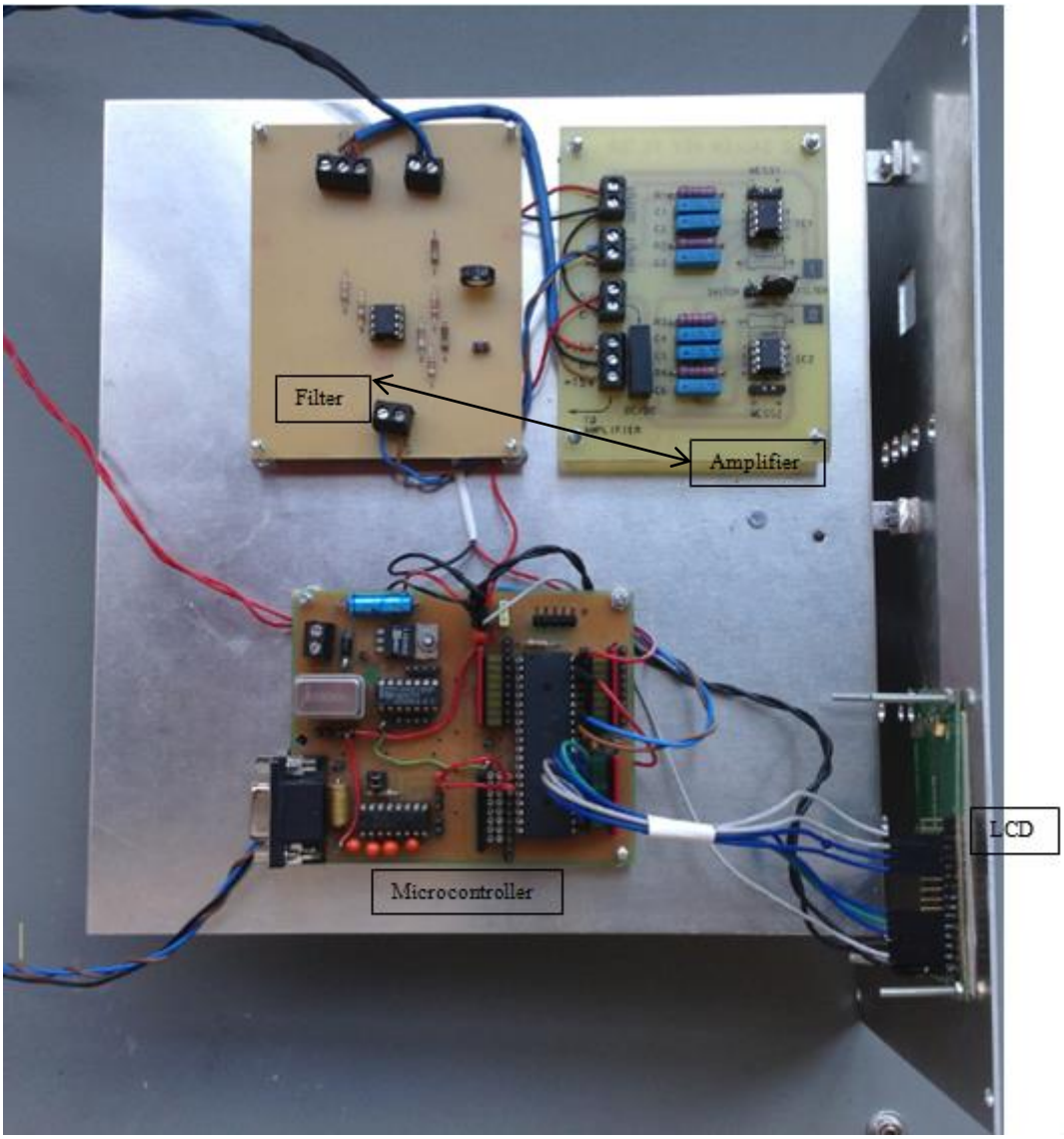


Figure 9: inside the box

Figure 10: box

Amplifier

In broad terms, an amplifier is an electronic device used to increase or decrease the strength of the signal fed into it. In this case the amplifier circuit board was already designed and manufactured. The input signal in the amplifier, which is the output signal coming from the transducer, has a range between -10V and 10V and the output signal has to be reduced in order to can be handled by the microcontroller. Thus, the signal range goes from 0V to 5V. For this purpose, an operational amplifier with a gain of ¼ is used and an offset voltage of +10V, which is calibrated using a trimmer, is provided at the positive input of the operational amplifier to work only with positive voltages.

Next, the theoretical calculations for the amplifier are shown:

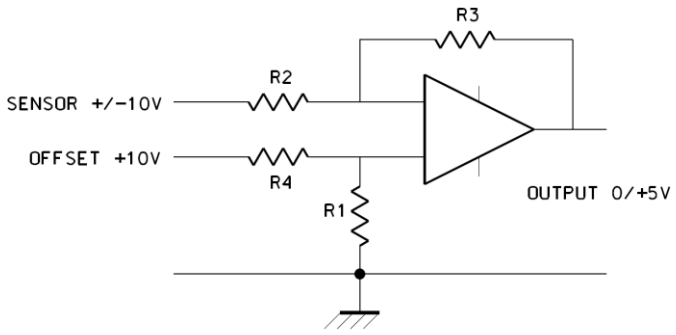


Figure 11: theoretical amplifier diagram

The output signal is defined by the next formula:

$$U_o = \frac{R_2}{R_1} \cdot (U_{i^+} - U_{i^-})$$

Considering: $R_2 = R_4$, $R_1 = R_3$, $U_{i^+} \overset{\text{def}}{=} offset$ and $U_{i^-} \overset{\text{def}}{=} signal$ and being the gain:

$$\frac{1}{4} = \frac{R_2}{R_1}$$

For $U_{i^-} = -10\ V$

$$U_o = \frac{1}{4} \cdot (10 - (-10)) = 5\ V$$

For $U_{i^-} = 10\ V$

$$U_o = \frac{1}{4} \cdot (10 - 10) = 0\ V$$

Consequentially, the values for the resistances are:

$$R_2 = R_4 = 1\ K\Omega$$

$$R_1 = R_3 = 4\ K\Omega$$

Filter

It has been needed to build a low-pass filter in order to let pass low-frequency signals and attenuate signals with frequencies higher than the cutoff frequency which is theoretically in this case 1 KHz. This filter consists of two Sallen- Key filters which can be used only one filter or both in cascade (in the filter layout, annex B, it is indicated with "1" the connection needed to be used only one filter and "2" when both are used). The switching of the two modes is performed by short step bars. Furthermore, in this board measuring pins are located in order to measure the output signals of the individual filters.

Following is shown the theoretical scheme of one filter and its calculations:

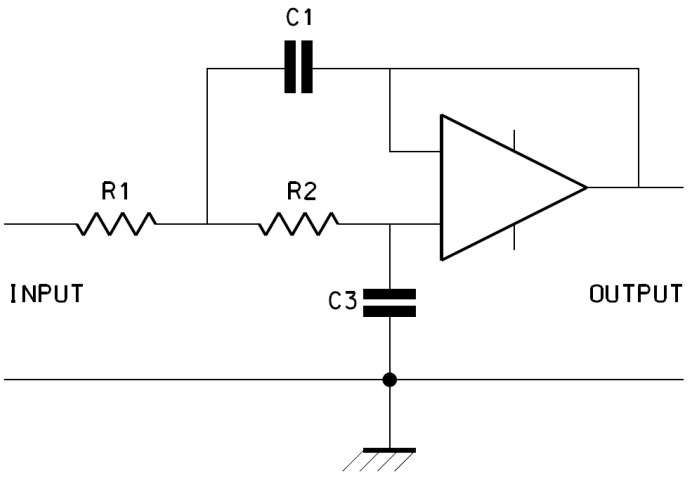


Figure 12: theoretical filter diagram

Theoretically, being the cutoff frequency $f_H = 1\ KHz$ and choosing $C_2 = 100\ nF$:

$$C_1 = 2 \cdot C_2 = 200\ nF$$

$$R_1 = R_2 = \frac{0{,}707}{2 \cdot \pi \cdot C_2 \cdot f_H} = 1125\ \Omega$$

In practice, using as values $R_1 = R_2 = 1{,}2\ K\Omega$

$$f_H = \frac{1}{2 \cdot \pi \cdot \sqrt{R_1 \cdot R_2 \cdot C_1 \cdot C_2}} \approx 938\ Hz$$

In Annex A is shown the whole process of manufacturing this filter, especially the printed circuit, which we decided to do manually in order to save time.

Microcontroller

The Atmega16 microcontroller manufactured by ATMEL is used in this project to process, convert and transmit the analogue signal received by the sensor to the PC. The microcontroller board was already manufactured. We could not read the program that was already in it. Therefore, we decided to remove all the connections and do them again and review and do the programming part again by ourselves.

We have joined the datasheet of this microcontroller in a pdf (click here to see them). The most important information is as below.

The main features of the atmega16 are:

• High-performance, Low-power Atmel ® AVR ® 8-bit Microcontroller

• Advanced RISC Architecture

– 131 Powerful Instructions – Most Single-clock Cycle Execution

– 32 x 8 General Purpose Working Registers

– Fully Static Operation

– Up to 16 MIPS Throughput at 16 MHz

– On-chip 2-cycle Multiplier

• High Endurance Non-volatile Memory segments

– 16 Kbytes of In-System Self-programmable Flash program memory

– 512 Bytes EEPROM

– 1 Kbyte Internal SRAM

– Write/Erase Cycles: 10,000 Flash/100,000 EEPROM

– Data retention: 20 years at 85°C/100 years at 25°C

– Optional Boot Code Section with Independent Lock Bits

In-System Programming by On-chip Boot Program

True Read-While-Write Operation

– Programming Lock for Software Security

• JTAG (IEEE std. 1149.1 Compliant) Interface

– Boundary-scan Capabilities According to the JTAG Standard

– Extensive On-chip Debug Support

– Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface

• Peripheral Features

– Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes

– One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode

– Real Time Counter with Separate Oscillator

– Four PWM Channels

– 8-channel, 10-bit ADC

8 Single-ended Channels

7 Differential Channels in TQFP Package Only

2 Differential Channels with Programmable Gain at 1x, 10x, or 200x

– Byte-oriented Two-wire Serial Interface

– Programmable Serial USART

– Master/Slave SPI Serial Interface

– Programmable Watchdog Timer with Separate On-chip Oscillator

– On-chip Analogue Comparator

• Special Microcontroller Features

– Power-on Reset and Programmable Brown-out Detection

– Internal Calibrated RC Oscillator

– External and Internal Interrupt Sources

– Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby

• I/O and Packages

– 32 Programmable I/O Lines

– 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF

• Operating Voltages

– 2.7V - 5.5V for ATmega16L

– 4.5V - 5.5V for ATmega16

• Speed Grades

– 0 - 8 MHz for ATmega16L

– 0 - 16 MHz for ATmega16

• Power Consumption @ 1 MHz, 3V, and 25°C for ATmega16L

– Active: 1.1 mA

– Idle Mode: 0.35 mA

– Power-down Mode: < 1 µA

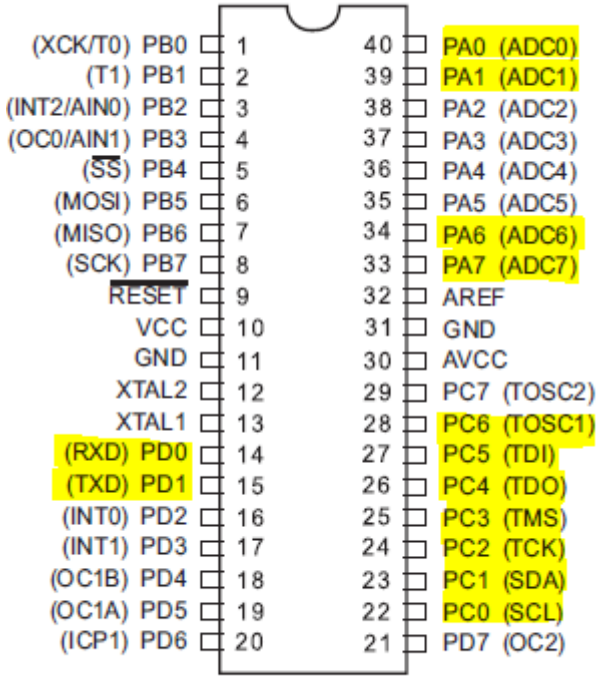


Figure 13: ATmega16 Pin Diagram

The pin configuration is shown above where pins used to connect the microcontroller to other devices are highlighted in yellow:

- Pins PD0 (USART input pin) and PD1 (USART output pin) allow us to transmit and read data between the serial port and the PC, respectively.
- Port A serves as the analog input to the A/D Converter. By pin PA0 the µC receives the analogue signal of the pressure; pin PA2 is used as an input for the feedback (*zero* comes from the mux to this pin and *comm*. cable is connected to 5V, so that when the mux is in home then in this pin there are 5 V); pin PA6 send a signal to come back home and pin PA7 sends a signal for stepping the mux.
- Pins used in C port are all for communication with the LCD display.

LCD display

A standard 2x15 lines LCD display has been used, whose features are:

- 5x8 dots with cursor
- 16 characters*2 lines display
- 4-bit or 8-bit MPU interfaces
- Built-in controller (ST7066 or equivalent)
- Display Mode & Backlight Variations
- ROHS Compilant

In the upper line will appear the pressure in mbar (I made a formula which transforms voltages in pressures. It appears in the programming code) and in the second line the channel in which the mux is in that moment will be shown (home will appear as "HOME").

Pin no.	Symbol	External connection	Function
1	Vss	Power supply	Signal ground for LCM
2	V_{DD}		Power supply for logic for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7~10	DB0~DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11~14	DB4~DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU
15	LED+	LED BKL power supply	Power supply for BKL
16	LED-		Power supply for BKL

Figure 14: LCD pin configuration

Used pins are highlighted in yellow. The pins going to the port C of the µC are number 4, 5, 6, 11, 12, 13, and 14. The three first pins are connected to ground and 5 volts.

SOFTWARE

AVR Atmega 16

The whole programming appears in Annex C: microcontroller programming, where there are a lot of explanations to facilitate the reading of the programming and the program used for that was AVR Studio 4.

One point which is important to be mentioned is that there was something wrong in the description of the signal *zero* coming out from the position transmitter because it was said in the specifications that this cable is only high if mux is in home position, but it is not true because the µC detected it high before being in home. For this reason, I decided that for stopping sending the signal home would meet two conditions:

1. Wait three seconds until checking if the switch is closed so that home detection would be powered with 5 volts (because *comm*. is connected to that voltage). I have chosen this period of time because would be the maximum time needed for going home. It would be the case of being in position 1. This condition was added for the problem previously mentioned.

2. Once that home detection is high, home pin stops being high, so that the µC stops sending home signal to the input of the optocoupler board.

In the program of the µC has been also included the formula which allows to transform voltage in pressure, whose values are shown in LabView. That voltage is the one after the amplifier, it means that goes from 0 V to 5 V. As the specifications of the pressure transducer indicate, the pressure has to be between -50 mbar and +50 mbar. The theoretical formula was easy to find out thanks to the linear link between both parameters:

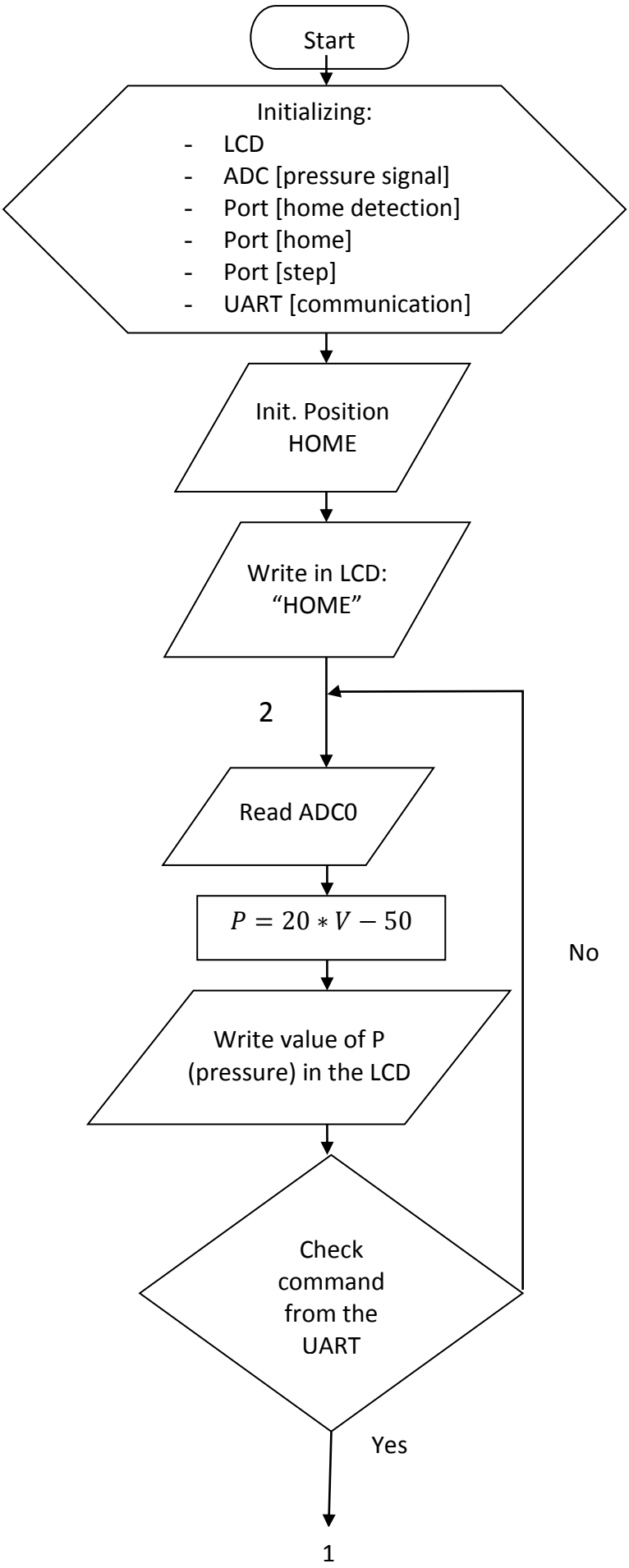
$P = 20 \cdot V - 50$, where the units for pressure (P) are mbar and for voltage (V) are volts.

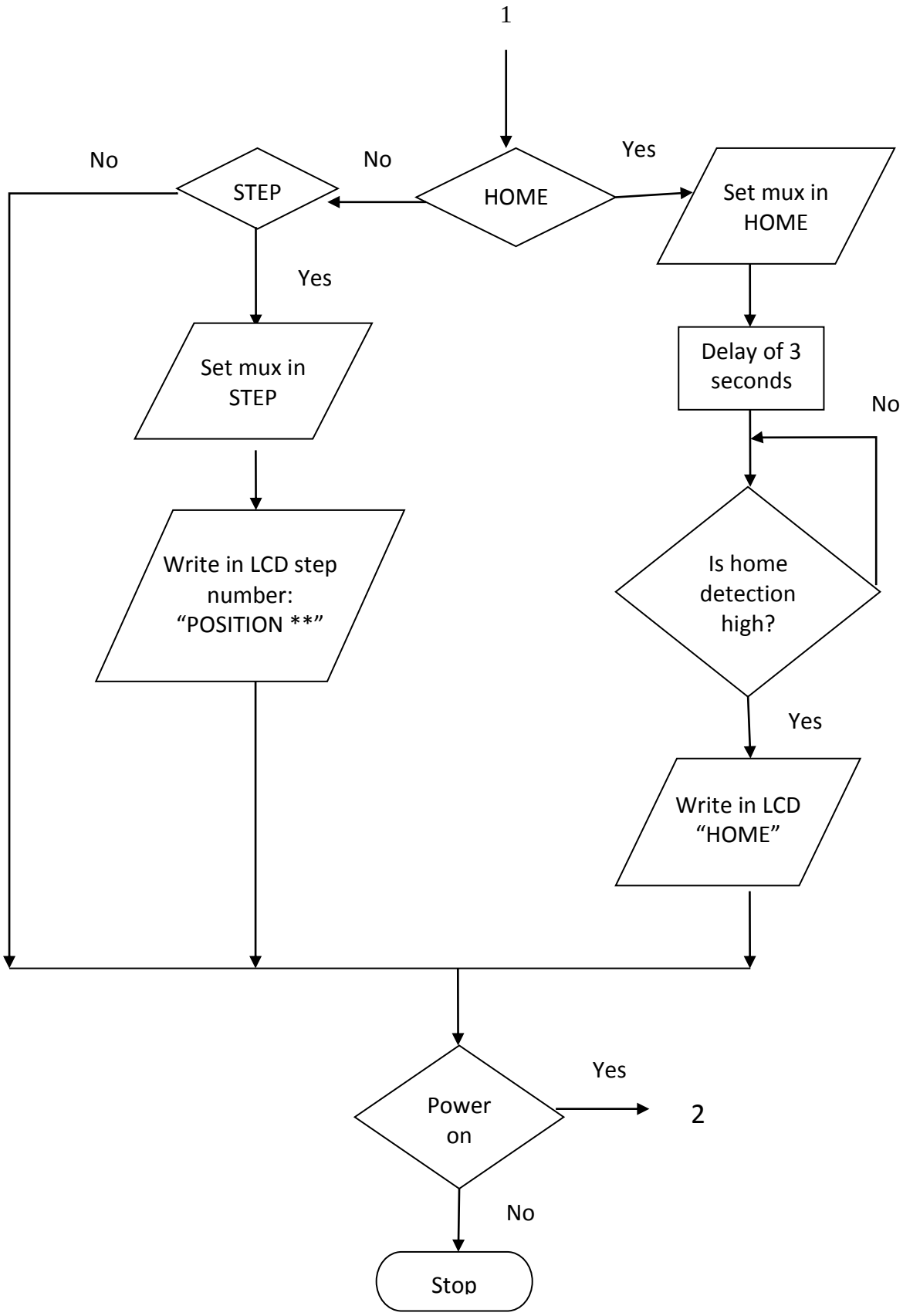
Based in a thesis of the same project made by other students, we got an empirical formula shown below these lines, which is used in the program:

$$P = 19{,}86 \cdot V - 50{,}917.$$

I have done a flow diagram to understand better how the microcontroller programming works. You can find it in the next page:

FLOW CHART


Start
Initializing:
- LCD
- ADC [pressure signal]
- Port [home detection]
- Port [home]
- Port [step]
- UART [communication]
Init. Position HOME
Write in LCD: "HOME"
2
Read ADC0
$P = 20 * V - 50$
Write value of P (pressure) in the LCD
Check command from the UART
No
Yes
1


1
HOME
No
STEP
No
Yes
Set mux in HOME
Yes
Set mux in STEP
Delay of 3 seconds
No
Write in LCD step number: "POSITION **"
Is home detection high?
Yes
Write in LCD "HOME"
Power on
Yes
2
No
Stop

LabView

Introduction

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is one programming tool developed by National Instruments which allows visualizing much more easily processes in graphical form on the computer screen and an overview of our program. This development in process visualization allows, as example explained by professor Georgi in his lessons in where we have assisted, the process of filling a container to no longer (or solely) be depicted by an analogue manometer, but also by a graphical representation of the container and the colored liquid within. The goal of that is to instantly visualize the status of the technical process saving time and making easy our work.

LabVIEW is not just a programming tool; it is a Program Development System, which contains an independent graphical programming language, called 'G'. One advantage of LabView is the possibility to be connected to other well known programming languages: for example, LabVIEW allows the insertion of C-Code.

Program

My colleague Moisés González made most of the LabView program, so that we decided that he will explain it in the report that he will deliver to our supervisor separately. It has not been included in this report due to a difference between us in relation to the delivery date.

CONCLUSION

This project has been a very positive experience for me because I have learned a lot of extra topics which were not related with energies at all. At the beginning I had big difficulties for manufacturing the filter and optocopler, for getting the electronic components needed, for finding out how get a feedback of home position and much more. Despite those obstacles, we overcame them successfully.

The other two difficulties were to make the communication between the microcontroller and the computer (with AVR Studio program) and between the microcontroller and LabView. After one month and a half, we finally achieved it.

The last steps have been the most complicated to accomplish: the programming of the microcontroller and LAbview.

In conclusion, the aim of this project has been successfully accomplished. I have acquired large knowledge about electronics and programming so what it has been a worthy experience for me.

REFERENCES

i. The handout of the lecture “LabView”. Author: Prof. Georgi.

ii. ATmega16 Datasheet, 2010.

iii. LabVIEW Technical Resources:
http://www.ni.com/labview/technical-resources/
http://perso.wanadoo.es/jovilve/tutoriales/016tutorlabview.pdf
LAbview help

iv. Microcontroller. Electronic resources:
http://www.engineersgarage.com/electronic-components/atmega16-microcontroller
http://home.iitk.ac.in/~sbanshi/robotics/tutorials/Introduction%20to%20Atmega%2016%20Microcontroller
www.avrtutorials.com

ANNEXES

Annex A: manufacturing of the printed circuit of the filter

In this section, I want to explain in detail how I made the printed circuit of the optocoupler and filter. I decided to make it manually in order to know first-hand the procedure and also I wanted to save time because if I had sent it to the electronics laboratory, three weeks more had been required till I the boards were finished.

There are different methods by which a printed circuit board can be created (PCB) for an electrical/electronic circuit and I chose the acid etching method.

The circuit diagram was printed with a laser printer in a glossy paper and this was fixed temporarily on an 80x100 mm fiberglass board whose glass thickness is 1,5 mm and copper thickness is 35 µm only in one side (the glossy side, with the circuit part printing on it, has to be facing the copper side). Next, I ironed the paper using an ordinary clothes iron. The amount of time this will take depends on the type of paper and ink used. In my case, ten minutes was needed. In the other side, where the components are located, is exactly the same procedure but in the non-copper side.

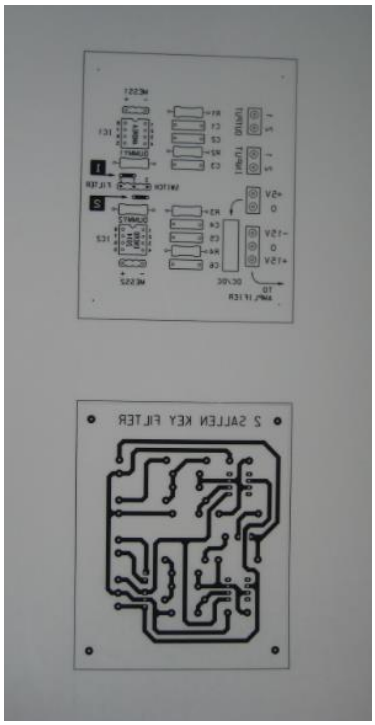


Figure 15: printed circuit diagram in a glossy paper

Figure 16: ironing the filter board

After ironing, I removed the paper and submerged the board in ferric chloride (acid) until that all unnecessary copper was etched away from the board (twenty minutes approximately).

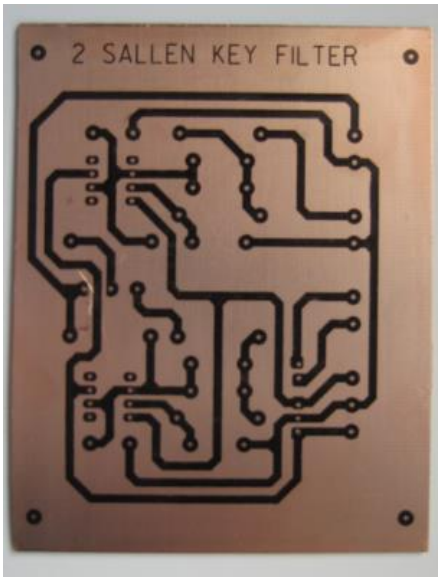


Figure 17: after ironing the filter board

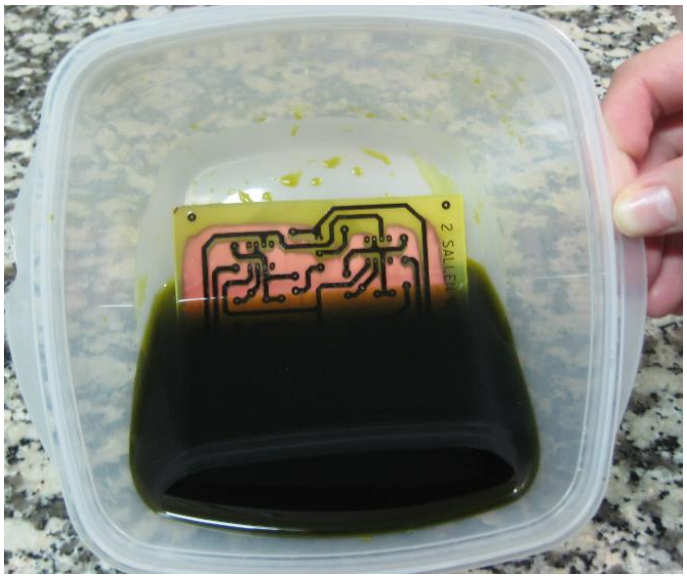


Figure 18: board submerged in ferric chloride

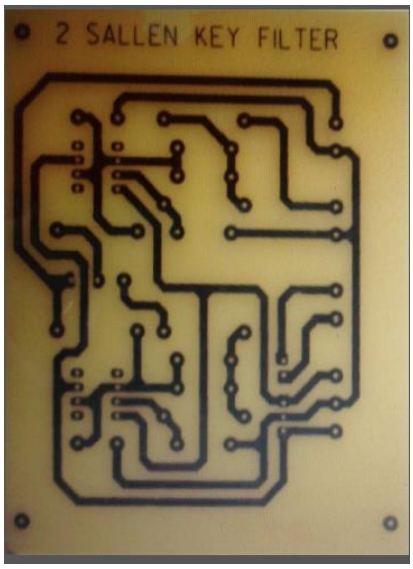


Figure 19: unnecessary copper removed

In order to remove the insulating drawing material used (toner in this case); I used acetone and sand paper (a fine one). Finally, I drilled to make holes and placed the components in the board.

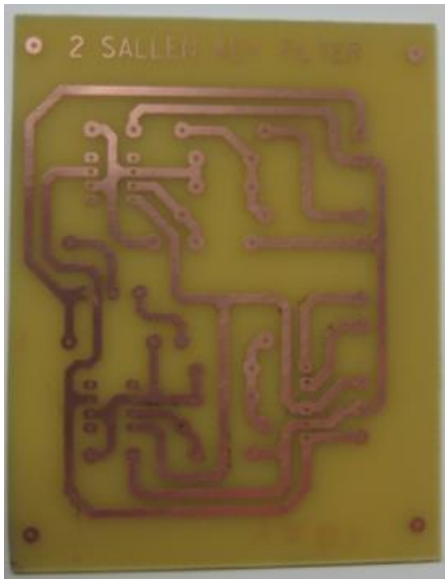


Figure 20: toner removed

Figure 21: drilling

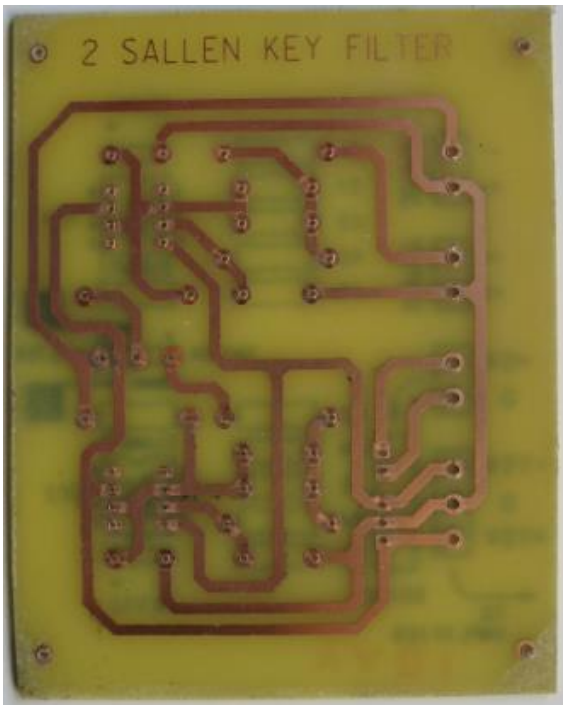


Figure 22: finished filter

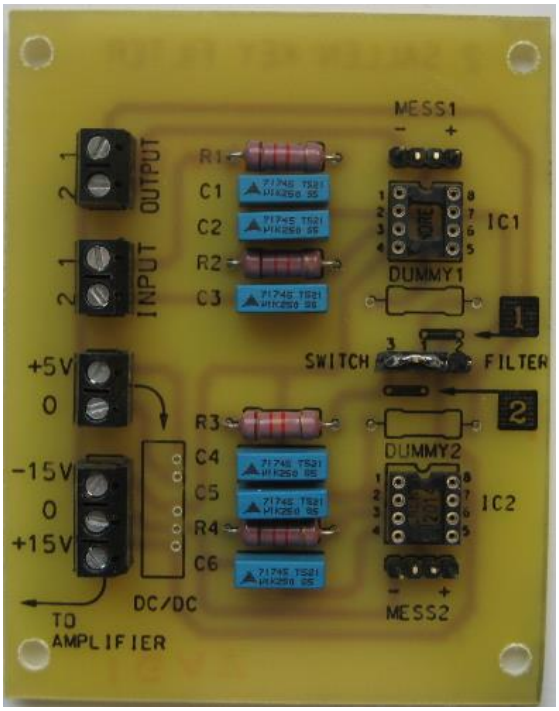


Figure 23: drilled board

Annex B: circuits diagram and layout of the boards manufactured by us

Amplifier

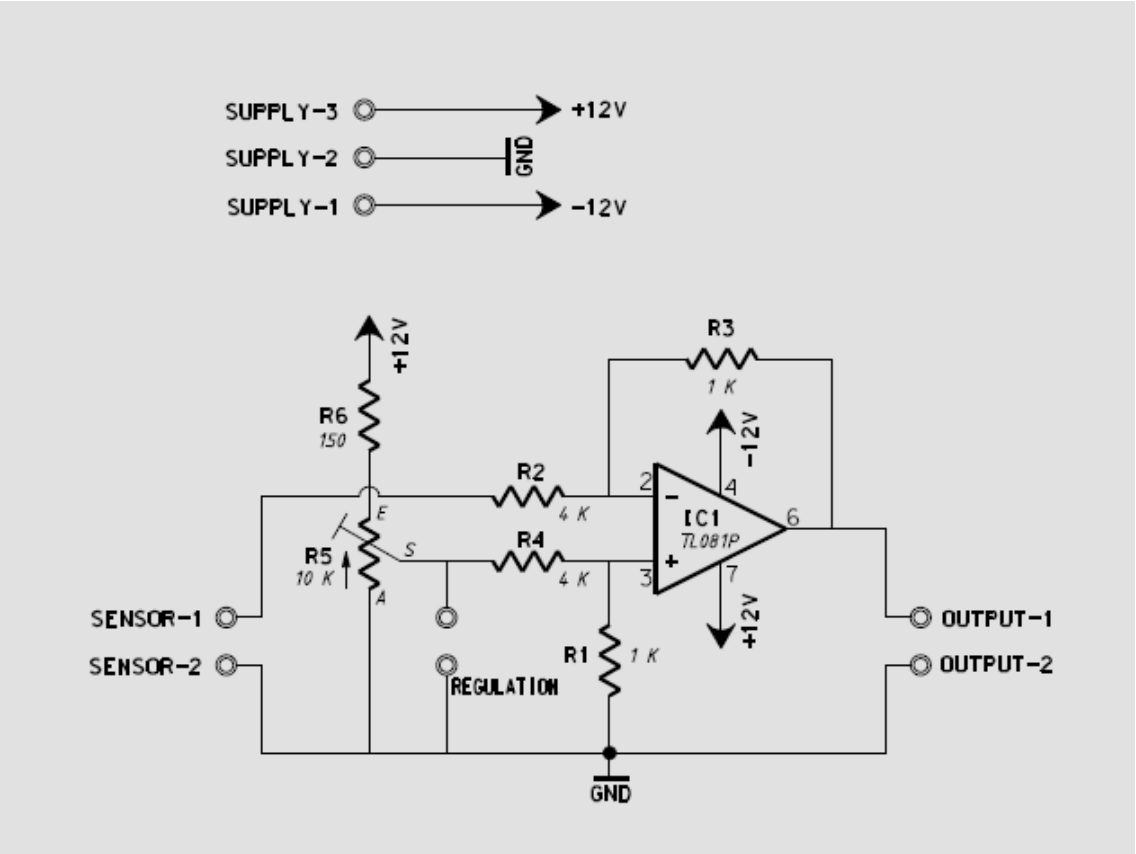


Figure 24: amplifier circuit diagram

Filter

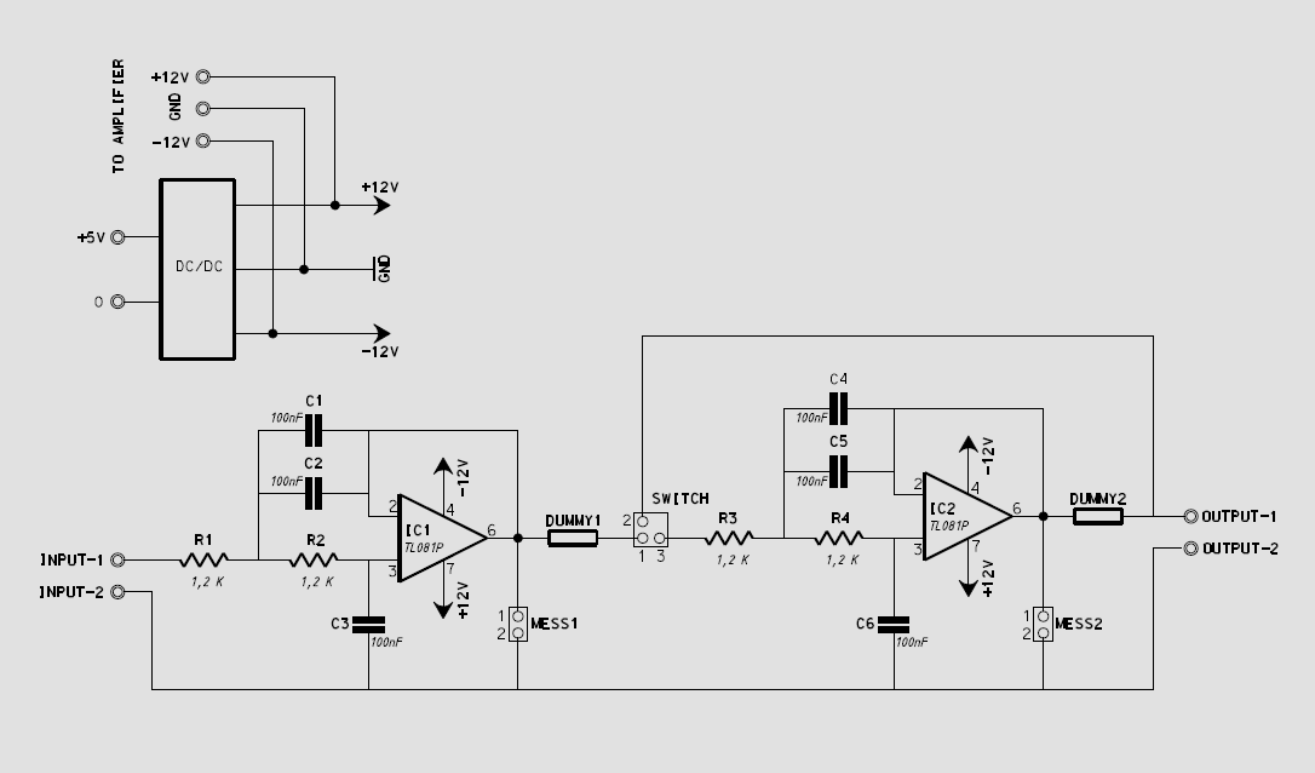


Figure 25: filter circuit diagram

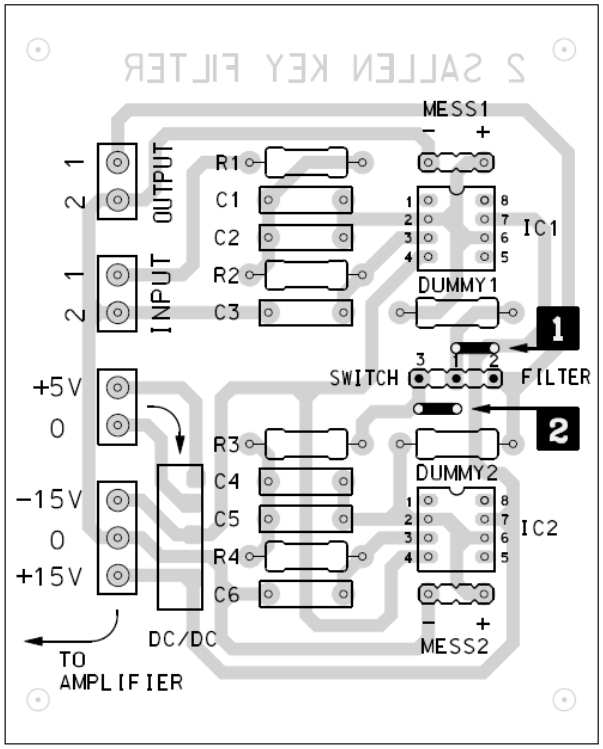


Figure 26: filter layout

Optocoupler

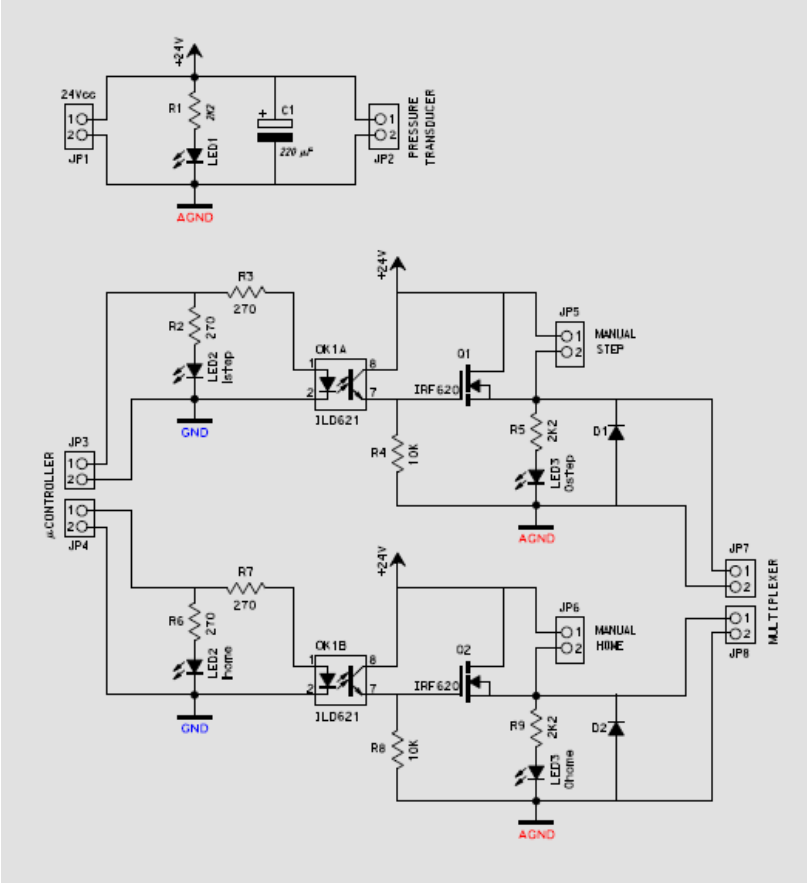


Figure 27: optocoupler circuit diagram

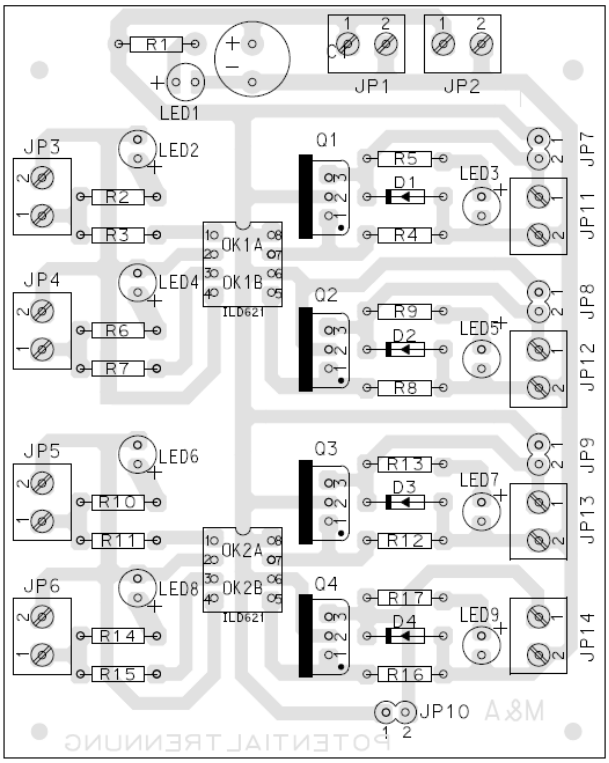


Figure 28: optocoupler layout

Annex C: microcontroller programming

Main program

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <avr/signal.h>
#include <avr/pgmspace.h>
#include <string.h>
#include <util/delay.h>
//-------------------------
#include "lcd.h"
#include "uart.h"
#include "ADC.h"
#include "Multiplexer.h"
//-------------------------
/* define CPU frequency in Mhz here if not defined in Makefile */
#ifndef F_CPU
#define F_CPU 8000000UL
#endif

int main (void)
{
 DDRA=0xF0;
 sei();
 Serial_Config();
 lcd_init(LCD_DISP_ON);              // initialize display, cursor off
 lcd_clrscr();
 lcd_puts("    TURBINE\n");
 lcd_puts("  RTPC Ver 0.1\n");
 lcd_time_delay();
 lcd_time_delay();
 init_ADC();
 lcd_clrscr();
 lcd_puts("Pressure  : \n");
 lcd_puts("Mux Select: ");
 read_ADC();
 mux_Home();
 //mux_Step();
do
 {
Rep:
        read_ADC();
        RXData_Labview();
        goto Rep;
 }
  while(0);
  return 0;
}
```

LCD.c subprogram

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "lcd.h"
#include<util/delay.h>
/*
** constants/macros
*/
#define DDR(x) (*(&x - 1))      /* address of data direction register of port
#if defined(__AVR_ATmega64__) || defined(__AVR_ATmega128__)
    /* on ATmega64/128 PINF is on port 0x00 and not 0x60 */
    #define PIN(x) ( &PORTF==&(x) ? _SFR_IO8(0x00) : (*(&x - 2)) )
#else
    #define PIN(x) (*(&x - 2))    /* address of input register of port x
#endif

#if LCD_IO_MODE
#define lcd_e_delay()   __asm__ __volatile__( "rjmp 1f\n 1:" );
#define lcd_e_high()    LCD_E_PORT  |=  _BV(LCD_E_PIN);
#define lcd_e_low()     LCD_E_PORT  &= ~_BV(LCD_E_PIN);
#define lcd_e_toggle()  toggle_e()
#define lcd_rw_high()   LCD_RW_PORT |=  _BV(LCD_RW_PIN)
#define lcd_rw_low()    LCD_RW_PORT &= ~_BV(LCD_RW_PIN)
#define lcd_rs_high()   LCD_RS_PORT |=  _BV(LCD_RS_PIN)
#define lcd_rs_low()    LCD_RS_PORT &= ~_BV(LCD_RS_PIN)
#endif

#if LCD_IO_MODE
#if LCD_LINES==1
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_4BIT_1LINE
#else
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_4BIT_2LINES
#endif
#else
#if LCD_LINES==1
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_8BIT_1LINE
#else
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_8BIT_2LINES
#endif
#endif

#if LCD_CONTROLLER_KS0073
#if LCD_LINES==4

#define KS0073_EXTENDED_FUNCTION_REGISTER_ON  0x2C   /* |0|010|0100 4-bit mod
#define KS0073_EXTENDED_FUNCTION_REGISTER_OFF 0x28   /* |0|000|1001 4 lines m
-------------------------------------------------------------------------------
#define KS0073_4LINES_MODE                    0x09   /* |0|001|0000 4-bit mod

#endif
#endif

/*
** function prototypes
*/
#if LCD_IO_MODE
static void toggle_e(void);
#endif

/*
** local functions
*/

void clear_2nd_row (void)
{
lcd_gotoxy(0,1);
lcd_puts("       \n");
lcd_time_delay();
}
//*** constant definitions for LCD
static const PROGMEM unsigned char copyRightChar[] =
{
    0x07, 0x08, 0x13, 0x14, 0x14, 0x13, 0x08, 0x07,
    0x00, 0x10, 0x08, 0x08, 0x08, 0x08, 0x10, 0x00
};
//***conver float number to string and display on LCD
void lcd_putn(float F)
{
    char wort[20];
    dtostrf(F,11,8,wort);
    lcd_puts(wort);
}
/*void Numout(float F)
{
     char wort[20];          //Number (Integer & Float) must be converted in A
     dtostrf(F,11,8,wort);
     ser_puts(wort);
}*/
void lcd_time_delay() {
     _delay_us(500000);
     _delay_us(500000);
}
```

```
/*************************************************************************
 delay loop for small accurate delays: 16-bit counter, 4 cycles/loop
*************************************************************************/
static inline void _delayFourCycles(unsigned int __count)
{
    if ( __count == 0 )
        __asm__ __volatile__( "rjmp 1f\n 1:" );    // 2 cycles
    else
        __asm__ __volatile__ (
            "1: sbiw %0,1" "\n\t"
            "brne 1b"                              // 4 cycles/loop
            : "=w" (__count)
            : "0" (__count)
           );
}
/*************************************************************************
delay for a minimum of <us> microseconds
the number of loops is calculated at compile-time from MCU clock frequency
*************************************************************************/
#define delay(us)  _delayFourCycles( ( ( 1*(XTAL/4000) )*us)/1000 )


#if LCD_IO_MODE
/* toggle Enable Pin to initiate write */
static void toggle_e(void)
{
    lcd_e_high();
    lcd_e_delay();
    lcd_e_low();
}
#endif
/*************************************************************************
Low-level function to write byte to LCD controller
Input:    data   byte to write to LCD
          rs     1: write data
                 0: write instruction
Returns:  none
*************************************************************************/
#if LCD_IO_MODE
static void lcd_write(uint8_t data,uint8_t rs)
{
    unsigned char dataBits ;


    if (rs) {   /* write data        (RS=1, RW=0) */
       lcd_rs_high();
 } else {    /* write instruction (RS=0, RW=0) */
    lcd_rs_low();
 }
 lcd_rw_low();

 if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) &&
 ( &LCD_DATA1_PORT == &LCD_DATA2_PORT ) &&
 ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
   && (LCD_DATA0_PIN == 0) && (LCD_DATA1_PIN == 1)
   && (LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3) )
 {
     /* configure data pins as output */
     DDR(LCD_DATA0_PORT) |= 0x0F;

     /* output high nibble first */
     dataBits = LCD_DATA0_PORT & 0xF0;
     LCD_DATA0_PORT = dataBits |((data>>4)&0x0F);
     lcd_e_toggle();

     /* output low nibble */
     LCD_DATA0_PORT = dataBits | (data&0x0F);
     lcd_e_toggle();

     /* all data pins high (inactive) */
     LCD_DATA0_PORT = dataBits | 0x0F;
 }
 else
 {
     /* configure data pins as output */
     DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
     DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
     DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
     DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);

     /* output high nibble first */
     LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
     LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
     LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
     LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
     if(data & 0x80) LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
     if(data & 0x40) LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
     if(data & 0x20) LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
     if(data & 0x10) LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
     lcd_e_toggle();
```

```
        LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
        LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
        LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
        LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
        if(data & 0x08) LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
        if(data & 0x04) LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
        if(data & 0x02) LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
        if(data & 0x01) LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
        lcd_e_toggle();

        /* all data pins high (inactive) */
        LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
        LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
        LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
        LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
    }
}
#else
#define lcd_write(d,rs) if (rs) *(volatile uint8_t*)(LCD_IO_DATA) = d;
 else *(volatile uint8_t*)(LCD_IO_FUNCTION) = d;
/* rs==0 -> write instruction to LCD_IO_FUNCTION */
/* rs==1 -> write data to LCD_IO_DATA */
#endif
/*************************************************************************
Low-level function to read byte from LCD controller
Input:    rs     1: read data
                 0: read busy flag / address counter
Returns:  byte read from LCD controller
*************************************************************************/
#if LCD_IO_MODE
static uint8_t lcd_read(uint8_t rs)
{
    uint8_t data;


    if (rs)
        lcd_rs_high();                       /* RS=1: read data      */
    else
        lcd_rs_low();                        /* RS=0: read busy flag */
    lcd_rw_high();                           /* RW=1  read mode      */

    if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT)
    && ( &LCD_DATA1_PORT == &LCD_DATA2_PORT ) &&
     ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
      && ( LCD_DATA0_PIN == 0 )&& (LCD_DATA1_PIN == 1) &&
       (LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3) )

    {
        DDR(LCD_DATA0_PORT) &= 0xF0;         /* configure data pins as input

        lcd_e_high();
        lcd_e_delay();
        data = PIN(LCD_DATA0_PORT) << 4;     /* read high nibble first */
        lcd_e_low();

        lcd_e_delay();                       /* Enable 500ns low       */

        lcd_e_high();
        lcd_e_delay();
        data |= PIN(LCD_DATA0_PORT)&0x0F;    /* read low nibble        */
        lcd_e_low();
    }
    else
    {
        /* configure data pins as input */
        DDR(LCD_DATA0_PORT) &= ~_BV(LCD_DATA0_PIN);
        DDR(LCD_DATA1_PORT) &= ~_BV(LCD_DATA1_PIN);
        DDR(LCD_DATA2_PORT) &= ~_BV(LCD_DATA2_PIN);
        DDR(LCD_DATA3_PORT) &= ~_BV(LCD_DATA3_PIN);

        /* read high nibble first */
        lcd_e_high();
        lcd_e_delay();
        data = 0;
        if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) ) data |= 0x10;
        if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) ) data |= 0x20;
        if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) ) data |= 0x40;
        if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) ) data |= 0x80;
        lcd_e_low();

        lcd_e_delay();                       /* Enable 500ns low       */

        /* read low nibble */
        lcd_e_high();
        lcd_e_delay();
        if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) ) data |= 0x01;
        if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) ) data |= 0x02;
        if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) ) data |= 0x04;
        if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) ) data |= 0x08;
        lcd_e_low();
    }
    return data;
}
```

```
#else
#define lcd_read(rs) (rs) ? *(volatile uint8_t*)(LCD_IO_DATA+LCD_IO_READ) :
 *(volatile uint8_t*)(LCD_IO_FUNCTION+LCD_IO_READ)
/* rs==0 -> read instruction from LCD_IO_FUNCTION */
/* rs==1 -> read data from LCD_IO_DATA */
#endif

/*************************************************************************
loops while lcd is busy, returns address counter
*************************************************************************/
static uint8_t lcd_waitbusy(void)

{
    register uint8_t c;

    /* wait until busy flag is cleared */
    while ( (c=lcd_read(0)) & (1<<LCD_BUSY)) {}

    /* the address counter is updated 4us after the busy flag is cleared */
    delay(2);

    /* now read the address counter */
    return (lcd_read(0));  // return address counter

}/* lcd_waitbusy */


/*************************************************************************
Move cursor to the start of next line or to the first line if the cursor
is already on the last line.
*************************************************************************/
static inline void lcd_newline(uint8_t pos)
{
    register uint8_t addressCounter;


#if LCD_LINES==1
    addressCounter = 0;
#endif
#if LCD_LINES==2
    if ( pos < (LCD_START_LINE2) )
        addressCounter = LCD_START_LINE2;
    else
        addressCounter = LCD_START_LINE1;
#endif
#if LCD_LINES==4
#if KS0073_4LINES_MODE
    if ( pos < LCD_START_LINE2 )
        addressCounter = LCD_START_LINE2;
    else if ( (pos >= LCD_START_LINE2) && (pos < LCD_START_LINE3) )
        addressCounter = LCD_START_LINE3;
    else if ( (pos >= LCD_START_LINE3) && (pos < LCD_START_LINE4) )
        addressCounter = LCD_START_LINE4;
    else
        addressCounter = LCD_START_LINE1;
#else
    if ( pos < LCD_START_LINE3 )
        addressCounter = LCD_START_LINE2;
    else if ( (pos >= LCD_START_LINE2) && (pos < LCD_START_LINE4) )
        addressCounter = LCD_START_LINE3;
    else if ( (pos >= LCD_START_LINE3) && (pos < LCD_START_LINE2) )
        addressCounter = LCD_START_LINE4;
    else
        addressCounter = LCD_START_LINE1;
#endif
#endif
    lcd_command((1<<LCD_DDRAM)+addressCounter);

}/* lcd_newline */

/*
** PUBLIC FUNCTIONS
*/

/*************************************************************************
Send LCD controller instruction command
Input:   instruction to send to LCD controller, see HD44780 data sheet
Returns: none
*************************************************************************/
void lcd_command(uint8_t cmd)
{
    lcd_waitbusy();
    lcd_write(cmd,0);
}

/*************************************************************************
Send data byte to LCD controller
Input:   data to send to LCD controller, see HD44780 data sheet
Returns: none
*************************************************************************/
void lcd_data(uint8_t data)
{
```

```
    lcd_waitbusy();
    lcd_write(data,1);
}

/*************************************************************************
Set cursor to specified position
Input:    x  horizontal position  (0: left most position)
          y  vertical position    (0: first line)
Returns:  none
*************************************************************************/
void lcd_gotoxy(uint8_t x, uint8_t y)
{
#if LCD_LINES==1
    lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
#endif
#if LCD_LINES==2
    if ( y==0 )
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
    else
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
#endif
#if LCD_LINES==4
    if ( y==0 )
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
    else if ( y==1)
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
    else if ( y==2)
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE3+x);
    else /* y==3 */
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE4+x);
#endif

}/* lcd_gotoxy */

/*************************************************************************
*************************************************************************/
int lcd_getxy(void)
{
    return lcd_waitbusy();
}
/*************************************************************************
Clear display and set cursor to home position
*************************************************************************/
void lcd_clrscr(void)
{
    lcd_command(1<<LCD_CLR);
}

/*************************************************************************
Set cursor to home position
*************************************************************************/
void lcd_home(void)
{
    lcd_command(1<<LCD_HOME);
}

/*************************************************************************
Display character at current cursor position
Input:    character to be displayed
Returns:  none
*************************************************************************/
void lcd_putc(char c)
{
    uint8_t pos;


    pos = lcd_waitbusy();   // read busy-flag and address counter
    if (c=='\n')
    {
        lcd_newline(pos);
    }
    else
    {
#if LCD_WRAP_LINES==1
#if LCD_LINES==1
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
        }
#elif LCD_LINES==2
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
        }else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH ){
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
        }
#elif LCD_LINES==4
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
        }else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE3,0);
        }else if ( pos == LCD_START_LINE3+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE4,0);
        }else if ( pos == LCD_START_LINE4+LCD_DISP_LENGTH ) {
```

```
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
        }
#endif
        lcd_waitbusy();
#endif
        lcd_write(c, 1);
    }

}/* lcd_putc */

/*************************************************************************
Display string without auto linefeed
Input:    string to be displayed
Returns:  none
*************************************************************************/
void lcd_puts(const char *s)
/* print string on lcd (no auto linefeed) */
{
    register char c;

    while ( (c = *s++) ) {
        lcd_putc(c);
    }

}/* lcd_puts */

/*************************************************************************
Display string from program memory without auto linefeed
Input:     string from program memory be be displayed
Returns:   none
*************************************************************************/
void lcd_puts_p(const char *progmem_s)
/* print string from program memory on lcd (no auto linefeed) */
{
    register char c;

    while ( (c = pgm_read_byte(progmem_s++)) ) {
        lcd_putc(c);
    }

}/* lcd_puts_p */

/*************************************************************************
Initialize display and select type of cursor
Input:    dispAttr LCD_DISP_OFF            display off
                   LCD_DISP_ON             display on, cursor off
                  LCD_DISP_ON             display on, cursor off
                  LCD_DISP_ON_CURSOR      display on, cursor on
                  LCD_DISP_CURSOR_BLINK   display on, cursor on flashing
Returns:  none
*************************************************************************/
void lcd_init(uint8_t dispAttr)
{
#if LCD_IO_MODE
    /*
     *  Initialize LCD to 4 bit I/O mode
     */

    if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT ==
    &LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
      && ( &LCD_RS_PORT == &LCD_DATA0_PORT) && ( &LCD_RW_PORT ==
      &LCD_DATA0_PORT) && (&LCD_E_PORT == &LCD_DATA0_PORT)
      && (LCD_DATA0_PIN == 0 ) && (LCD_DATA1_PIN == 1) &&
      (LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3)
      && (LCD_RS_PIN == 4 ) && (LCD_RW_PIN == 5) && (LCD_E_PIN == 6 ) )
    {
        /* configure all port bits as output (all LCD lines on same port) */
        DDR(LCD_DATA0_PORT) |= 0xFF;
    }
    else if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) &&
    ( &LCD_DATA1_PORT == &LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT ==
    &LCD_DATA3_PORT )
           && (LCD_DATA0_PIN == 0 ) && (LCD_DATA1_PIN == 1) &&
           (LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3) )
    {
        /* configure all port bits as output (all LCD data lines on same por
        but control lines on different ports) */
        DDR(LCD_DATA0_PORT) |= 0x0F;
        DDR(LCD_RS_PORT)    |= _BV(LCD_RS_PIN);
        DDR(LCD_RW_PORT)    |= _BV(LCD_RW_PIN);
        DDR(LCD_E_PORT)     |= _BV(LCD_E_PIN);
    }
    else
    {
        /* configure all port bits as output (LCD data and control lines on
        different ports */
        DDR(LCD_RS_PORT)    |= _BV(LCD_RS_PIN);
        DDR(LCD_RW_PORT)    |= _BV(LCD_RW_PIN);
        DDR(LCD_E_PORT)     |= _BV(LCD_E_PIN);
        DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
        DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
```

```
        DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
        DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);
    }
    delay(16000);        /* wait 16ms or more after power-on       */

    /* initial write to lcd is 8bit */
    LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);  // _BV(LCD_FUNCTION)>>4;
    LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);  // _BV(LCD_FUNCTION_8BIT)>>4;
    lcd_e_toggle();
    delay(4992);         /* delay, busy flag can't be checked here */

    /* repeat last command */
    lcd_e_toggle();
    delay(64);           /* delay, busy flag can't be checked here */

    /* repeat last command a third time */
    lcd_e_toggle();
    delay(64);           /* delay, busy flag can't be checked here */

    /* now configure for 4bit mode */
    LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);   // LCD_FUNCTION_4BIT_1LINE>>4
    lcd_e_toggle();
    delay(64);           /* some displays need this additional delay */

    /* from now the LCD only accepts 4 bit I/O, we can use lcd_command() */
#else
    /*
     * Initialize LCD to 8 bit memory mapped mode
     */

    /* enable external SRAM (memory mapped lcd) and one wait state */
    MCUCR = _BV(SRE) | _BV(SRW);

    /* reset LCD */
    delay(16000);                           /* wait 16ms after power-on     *
    lcd_write(LCD_FUNCTION_8BIT_1LINE,0);   /* function set: 8bit interface *
    delay(4992);                            /* wait 5ms                     *
    lcd_write(LCD_FUNCTION_8BIT_1LINE,0);   /* function set: 8bit interface *
    delay(64);                              /* wait 64us                    *
    lcd_write(LCD_FUNCTION_8BIT_1LINE,0);   /* function set: 8bit interface *
    delay(64);                              /* wait 64us                    *
#endif

#if KS0073_4LINES_MODE
    /* Display with KS0073 controller requires special commands for enabling
    | 4 line mode */
    lcd_command(KS0073_EXTENDED_FUNCTION_REGISTER_ON);
    lcd_command(KS0073_4LINES_MODE);
    lcd_command(KS0073_EXTENDED_FUNCTION_REGISTER_OFF);
#else
    lcd_command(LCD_FUNCTION_DEFAULT);      /* function set: display lines  *
#endif
    lcd_command(LCD_DISP_OFF);              /* display off                  *
    lcd_clrscr();                           /* display clear                *
    lcd_command(LCD_MODE_DEFAULT);          /* set entry mode               *
    lcd_command(dispAttr);                  /* display/cursor control       *

}/* lcd_init */
```

LCD.h subprogram

```
#ifndef LCD_H
#define LCD_H

#if (__GNUC__ * 100 + __GNUC_MINOR__) < 303
#error"This library requires AVR-GCC 3.3 or later,update to newer AVR-GCC com
#endif

#include <inttypes.h>
#include <avr/pgmspace.h>

/**
 *  @name  Definitions for MCU Clock Frequency
 *  Adapt the MCU clock frequency in Hz to your target.
 */
#define XTAL 8000000              /**< clock frequency in Hz, used to calcula


/**
 * @name  Definition for LCD controller type
 * Use 0 for HD44780 controller, change to 1 for displays with KS0073 control
 */
#define LCD_CONTROLLER_KS0073 1  /**< Use 0 for HD44780 controller, 1 for KS0

/**
 *  @name  Definitions for Display Size
 *  Change these definitions to adapt setting to your display
 */
#define LCD_LINES           4     /**< number of visible lines of the display
#define LCD_DISP_LENGTH    20     /**< visibles characters per line of the di
#define LCD_LINE_LENGTH  0x13     /**< internal line length of the display
#define LCD_START_LINE1  0x00     /**< DDRAM address of first char of line 1
#define LCD_START_LINE2  0x40     /**< DDRAM address of first char of line 2
#define LCD_START_LINE3  0x14     /**< DDRAM address of first char of line 3
#define LCD_START_LINE4  0x54     /**< DDRAM address of first char of line 4
#define LCD_WRAP_LINES      1     /**< 0: no wrap, 1: wrap at end of visibile


#define LCD_IO_MODE      1         /**< 0: memory mapped mode, 1: IO port mod
#if LCD_IO_MODE
/**
 *  @name Definitions for 4-bit IO mode
 *  Change LCD_PORT if you want to use a different port for the LCD pins.
 *
 *  The four LCD data lines and the three control lines RS, RW, E can be on t
 *  same port or on different ports.
 *  Change LCD_RS_PORT, LCD_RW_PORT, LCD_E_PORT if you want the control lines
 *  different ports.
 *
 *  Normally the four data lines should be mapped to bit 0..3 on one port, bu
 *  is possible to connect these data lines in different order or even on dif
 *  ports by adapting the LCD_DATAx_PORT and LCD_DATAx_PIN definitions.
 *
 */
#define LCD_PORT         PORTC        /**< port for the LCD lines   */
#define LCD_DATA0_PORT   LCD_PORT     /**< port for 4bit data bit 0 */
#define LCD_DATA1_PORT   LCD_PORT     /**< port for 4bit data bit 1 */
#define LCD_DATA2_PORT   LCD_PORT     /**< port for 4bit data bit 2 */
#define LCD_DATA3_PORT   LCD_PORT     /**< port for 4bit data bit 3 */
#define LCD_DATA0_PIN    0            /**< pin for 4bit data bit 0  */
#define LCD_DATA1_PIN    1            /**< pin for 4bit data bit 1  */
#define LCD_DATA2_PIN    2            /**< pin for 4bit data bit 2  */
#define LCD_DATA3_PIN    3            /**< pin for 4bit data bit 3  */
#define LCD_RS_PORT      LCD_PORT     /**< port for RS line         */
#define LCD_RS_PIN       4            /**< pin  for RS line         */
#define LCD_RW_PORT      LCD_PORT     /**< port for RW line         */
#define LCD_RW_PIN       5            /**< pin  for RW line         */
#define LCD_E_PORT       LCD_PORT     /**< port for Enable line     */
#define LCD_E_PIN        6            /**< pin  for Enable line     */

#define LCD_LED       (1 << PC7)           // controling the back light of th
#define LCD_Led_On()  PORTC |=  LCD_LED    //
#define LCD_Led_Off() PORTC &= ~LCD_LED

#elif defined(__AVR_AT90S4414__) || defined(__AVR_AT90S8515__) || defined(__A
      defined(__AVR_ATmega8515__)|| defined(__AVR_ATmega103__) || defined(__A
      defined(__AVR_ATmega161__) || defined(__AVR_ATmega162__) || defined(__A
/*
 *  memory mapped mode is only supported when the device has an external data
 */
#define LCD_IO_DATA      0xC000    /* A15=E=1, A14=RS=1                */
#define LCD_IO_FUNCTION  0x8000    /* A15=E=1, A14=RS=0                */
#define LCD_IO_READ      0x0100    /* A8 =R/W=1 (R/W: 1=Read, 0=Write  */
#else
#error "external data memory interface not available for this device, use 4-b

#endif

/**
 *  @name Definitions for LCD command instructions
 *  The constants define the various LCD controller instructions which can be
 *  function lcd_command(), see HD44780 data sheet for a complete description
 */
```

```
/* instruction register bit positions, see HD44780U data sheet */
#define LCD_CLR               0      /* DB0: clear display                  *
#define LCD_HOME              1      /* DB1: return to home position        *
#define LCD_ENTRY_MODE        2      /* DB2: set entry mode                 *
#define LCD_ENTRY_INC         1      /*   DB1: 1=increment, 0=decrement     *
#define LCD_ENTRY_SHIFT       0      /*   DB2: 1=display shift on           *
#define LCD_ON                3      /* DB3: turn lcd/cursor on             *
#define LCD_ON_DISPLAY        2      /*   DB2: turn display on              *
#define LCD_ON_CURSOR         1      /*   DB1: turn cursor on               *
#define LCD_ON_BLINK          0      /*     DB0: blinking cursor ?          *
#define LCD_MOVE              4      /* DB4: move cursor/display            *
#define LCD_MOVE_DISP         3      /*   DB3: move display (0-> cursor) ?  *
#define LCD_MOVE_RIGHT        2      /*   DB2: move right (0-> left) ?      *
#define LCD_FUNCTION          5      /* DB5: function set                   *
#define LCD_FUNCTION_8BIT     4      /*   DB4: set 8BIT mode (0->4BIT mode) *
#define LCD_FUNCTION_2LINES   3      /*   DB3: two lines (0->one line)      *
#define LCD_FUNCTION_10DOTS   2      /*   DB2: 5x10 font (0->5x7 font)      *
#define LCD_CGRAM             6      /* DB6: set CG RAM address             *
#define LCD_DDRAM             7      /* DB7: set DD RAM address             *
#define LCD_BUSY              7      /* DB7: LCD is busy                    *

/* set entry mode: display shift on/off, dec/inc cursor move direction */
#define LCD_ENTRY_DEC         0x04   /* display shift off, dec cursor move
#define LCD_ENTRY_DEC_SHIFT   0x05   /* display shift on,  dec cursor move
#define LCD_ENTRY_INC         0x06   /* display shift off, inc cursor move
#define LCD_ENTRY_INC_SHIFT   0x07   /* display shift on,  inc cursor move

/* display on/off, cursor on/off, blinking char at cursor position */
#define LCD_DISP_OFF          0x08   /* display off
#define LCD_DISP_ON           0x0C   /* display on, cursor off
#define LCD_DISP_ON_BLINK     0x0D   /* display on, cursor off, blink char
#define LCD_DISP_ON_CURSOR    0x0E   /* display on, cursor on
#define LCD_DISP_ON_CURSOR_BLINK 0x0F   /* display on, cursor on, blink char

/* move cursor/shift display */
#define LCD_MOVE_CURSOR_LEFT  0x10   /* move cursor left  (decrement)
#define LCD_MOVE_CURSOR_RIGHT 0x14   /* move cursor right (increment)
#define LCD_MOVE_DISP_LEFT    0x18   /* shift display left
#define LCD_MOVE_DISP_RIGHT   0x1C   /* shift display right

/* function set: set interface data length and number of display lines */
#define LCD_FUNCTION_4BIT_1LINE  0x20   /* 4-bit interface, single line, 5x7
#define LCD_FUNCTION_4BIT_2LINES 0x28   /* 4-bit interface, dual line,   5x7
#define LCD_FUNCTION_8BIT_1LINE  0x30   /* 8-bit interface, single line, 5x7
#define LCD_FUNCTION_8BIT_2LINES 0x38   /* 8-bit interface, dual line,   5x7

#define LCD_MODE_DEFAULT     ((1<<LCD_ENTRY_MODE) | (1<<LCD_ENTRY_INC) )

/**
 *  @name Functions
 */

/**
 @brief    Initialize display and select type of cursor
 @param    dispAttr \b LCD_DISP_OFF display off\n
                    \b LCD_DISP_ON display on, cursor off\n
                    \b LCD_DISP_ON_CURSOR display on, cursor on\n
                    \b LCD_DISP_ON_CURSOR_BLINK display on, cursor on flashin
 @return  none
*/
extern void lcd_init(uint8_t dispAttr);

/**
 @brief    Clear display and set cursor to home position
 @param    void
 @return   none
*/
extern void lcd_clrscr(void);

/**
 @brief    Set cursor to home position
 @param    void
 @return   none
*/
extern void lcd_home(void);

/**
 @brief    Set cursor to specified position

 @param    x horizontal position\n (0: left most position)
 @param    y vertical position\n   (0: first line)
 @return   none
*/
extern void lcd_gotoxy(uint8_t x, uint8_t y);

/**
 @brief    Display character at current cursor position
 @param    c character to be displayed
 @return   none
*/
extern void lcd_putc(char c);
```

```
/**
 @brief    Display string without auto linefeed
 @param    s string to be displayed
 @return   none
*/
extern void lcd_puts(const char *s);

/**
 @brief    Display string from program memory without auto linefeed
 @param    s string from program memory be be displayed
 @return   none
 @see      lcd_puts_P
*/
extern void lcd_puts_p(const char *progmem_s);

/**
 @brief    Send LCD controller instruction command
 @param    cmd instruction to send to LCD controller, see HD44780 data sheet
 @return   none
*/
extern void lcd_command(uint8_t cmd);

/**
 @brief    Send data byte to LCD controller

 Similar to lcd_putc(), but without interpreting LF
 @param    data byte to send to LCD controller, see HD44780 data sheet
 @return   none
*/
extern void lcd_data(uint8_t data);

/**
 @brief macros for automatically storing string constant in program memory
*/
#define lcd_puts_P(__s)         lcd_puts_p(PSTR(__s))

/*@}*/
#endif //LCD_H
```

Uart.c subprogram

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <avr/pgmspace.h>
#include "uart.h"
#include <stdarg.h>
#include <string.h>

//#include "mydefs.h"
//#include "serial.h"
//#include "timer.h"

volatile unsigned char rbuf[RBUFLEN]; //Ringpuffer
volatile unsigned char rcnt, rpos;
volatile unsigned char recbuf;
volatile unsigned char busy;

/*
 *  constants and macros
 */

/* size of RX/TX buffers */
#define UART_RX_BUFFER_MASK ( UART_RX_BUFFER_SIZE - 1)
#define UART_TX_BUFFER_MASK ( UART_TX_BUFFER_SIZE - 1)

#if ( UART_RX_BUFFER_SIZE & UART_RX_BUFFER_MASK )
#error RX buffer size is not a power of 2
#endif
#if ( UART_TX_BUFFER_SIZE & UART_TX_BUFFER_MASK )
#error TX buffer size is not a power of 2
#endif

#if defined(__AVR_AT90S2313__) \
 || defined(__AVR_AT90S4414__) || defined(__AVR_AT90S4434__) \
 || defined(__AVR_AT90S8515__) || defined(__AVR_AT90S8535__) \
 || defined(__AVR_ATmega103__)
 /* old AVR classic or ATmega103 with one UART */
 #define AT90_UART
 #define UART0_RECEIVE_INTERRUPT   SIG_UART_RECV
 #define UART0_TRANSMIT_INTERRUPT  SIG_UART_DATA
 #define UART0_STATUS   USR
 #define UART0_CONTROL  UCR
 #define UART0_DATA     UDR
 #define UART0_UDRIE    UDRIE
#elif defined(__AVR_AT90S2333__) || defined(__AVR_AT90S4433__)
 /* old AVR classic with one UART */

 #define AT90_UART
 #define UART0_RECEIVE_INTERRUPT   SIG_UART_RECV
 #define UART0_TRANSMIT_INTERRUPT  SIG_UART_DATA
 #define UART0_STATUS   UCSRA
 #define UART0_CONTROL  UCSRB
 #define UART0_DATA     UDR
 #define UART0_UDRIE    UDRIE
#elif  defined(__AVR_ATmega8__)  || defined(__AVR_ATmega16__) ||
 defined(__AVR_ATmega32__) \
  || defined(__AVR_ATmega8515__) || defined(__AVR_ATmega8535__) \
  || defined(__AVR_ATmega323__)
  /* ATmega with one USART */
 #define ATMEGA_USART
 #define UART0_RECEIVE_INTERRUPT   SIG_UART_RECV
 #define UART0_TRANSMIT_INTERRUPT  SIG_UART_DATA
 #define UART0_STATUS   UCSRA
 #define UART0_CONTROL  UCSRB
 #define UART0_DATA     UDR
 #define UART0_UDRIE    UDRIE
#elif defined(__AVR_ATmega163__)
  /* ATmega163 with one UART */
 #define ATMEGA_UART
 #define UART0_RECEIVE_INTERRUPT   SIG_UART_RECV
 #define UART0_TRANSMIT_INTERRUPT  SIG_UART_DATA
 #define UART0_STATUS   UCSRA
 #define UART0_CONTROL  UCSRB
 #define UART0_DATA     UDR
 #define UART0_UDRIE    UDRIE
#elif defined(__AVR_ATmega162__)
 /* ATmega with two USART */
 #define ATMEGA_USART0
 #define ATMEGA_USART1
 #define UART0_RECEIVE_INTERRUPT   SIG_USART0_RECV
 #define UART1_RECEIVE_INTERRUPT   SIG_USART1_RECV
 #define UART0_TRANSMIT_INTERRUPT  SIG_USART0_DATA
 #define UART1_TRANSMIT_INTERRUPT  SIG_USART1_DATA
 #define UART0_STATUS   UCSR0A
 #define UART0_CONTROL  UCSR0B
 #define UART0_DATA     UDR0
 #define UART0_UDRIE    UDRIE0
 #define UART1_STATUS   UCSR1A
 #define UART1_CONTROL  UCSR1B
 #define UART1_DATA     UDR1
 #define UART1_UDRIE    UDRIE1
#elif defined(__AVR_ATmega64__) || defined(__AVR_ATmega128__)
 /* ATmega with two USART */
```

```
 #define ATMEGA_USART0
 #define ATMEGA_USART1
 #define UART0_RECEIVE_INTERRUPT   SIG_UART0_RECV
 #define UART1_RECEIVE_INTERRUPT   SIG_UART1_RECV
 #define UART0_TRANSMIT_INTERRUPT  SIG_UART0_DATA
 #define UART1_TRANSMIT_INTERRUPT  SIG_UART1_DATA
 #define UART0_STATUS   UCSR0A
 #define UART0_CONTROL  UCSR0B
 #define UART0_DATA     UDR0
 #define UART0_UDRIE    UDRIE0
 #define UART1_STATUS   UCSR1A
 #define UART1_CONTROL  UCSR1B
 #define UART1_DATA     UDR1
 #define UART1_UDRIE    UDRIE1
#elif defined(__AVR_ATmega161__)
 /* ATmega with UART */
 #error "AVR ATmega161 currently not supported by this libaray !"
#elif defined(__AVR_ATmega169__)
 /* ATmega with one USART */
 #define ATMEGA_USART
 #define UART0_RECEIVE_INTERRUPT   SIG_USART_RECV
 #define UART0_TRANSMIT_INTERRUPT  SIG_USART_DATA
 #define UART0_STATUS   UCSRA
 #define UART0_CONTROL  UCSRB
 #define UART0_DATA     UDR
 #define UART0_UDRIE    UDRIE
#elif defined(__AVR_ATmega48__) ||defined(__AVR_ATmega88__) ||
defined(__AVR_ATmega168__)
 #define ATMEGA_USART0
 #define UART0_RECEIVE_INTERRUPT   SIG_USART_RECV
 #define UART0_TRANSMIT_INTERRUPT  SIG_USART_DATA
 #define UART0_STATUS   UCSR0A
 #define UART0_CONTROL  UCSR0B
 #define UART0_DATA     UDR0
 #define UART0_UDRIE    UDRIE0
#elif defined(__AVR_ATtiny2313__)
 #define ATMEGA_USART
 #define UART0_RECEIVE_INTERRUPT   SIG_USART0_RX
 #define UART0_TRANSMIT_INTERRUPT  SIG_USART0_UDRE
 #define UART0_STATUS   UCSRA
 #define UART0_CONTROL  UCSRB
 #define UART0_DATA     UDR
 #define UART0_UDRIE    UDRIE
#else
 #error "no UART definition for MCU available"
#endif
/*
 *  module global variables
 */
static volatile unsigned char UART_TxBuf[UART_TX_BUFFER_SIZE];
static volatile unsigned char UART_RxBuf[UART_RX_BUFFER_SIZE];
static volatile unsigned char UART_TxHead;
static volatile unsigned char UART_TxTail;
static volatile unsigned char UART_RxHead;
static volatile unsigned char UART_RxTail;
static volatile unsigned char UART_LastRxError;

#if defined( ATMEGA_USART1 )
static volatile unsigned char UART1_TxBuf[UART_TX_BUFFER_SIZE];
static volatile unsigned char UART1_RxBuf[UART_RX_BUFFER_SIZE];
static volatile unsigned char UART1_TxHead;
static volatile unsigned char UART1_TxTail;
static volatile unsigned char UART1_RxHead;
static volatile unsigned char UART1_RxTail;
static volatile unsigned char UART1_LastRxError;
#endif

SIGNAL(UART0_RECEIVE_INTERRUPT)
/*************************************************************************
Function: UART Receive Complete interrupt
Purpose:  called when the UART has received a character
*************************************************************************/
{
    unsigned char tmphead;
    unsigned char data;
    unsigned char usr;
    unsigned char lastRxError;

    /* read UART status register and UART data register */
    usr  = UART0_STATUS;
    data = UART0_DATA;

    /* */
#if defined( AT90_UART )
    lastRxError = (usr & (_BV(FE)|_BV(DOR)) );
#elif defined( ATMEGA_USART )
    lastRxError = (usr & (_BV(FE)|_BV(DOR)) );
#elif defined( ATMEGA_USART0 )
    lastRxError = (usr & (_BV(FE0)|_BV(DOR0)) );
#elif defined ( ATMEGA_UART )
    lastRxError = (usr & (_BV(FE)|_BV(DOR)) );
#endif
```

```
    /* calculate buffer index */
    tmphead = ( UART_RxHead + 1) & UART_RX_BUFFER_MASK;

    if ( tmphead == UART_RxTail ) {
        /* error: receive buffer overflow */
        lastRxError = UART_BUFFER_OVERFLOW >> 8;
    }else{
        /* store new index */
        UART_RxHead = tmphead;
        /* store received data in buffer */
        UART_RxBuf[tmphead] = data;
    }
    UART_LastRxError = lastRxError;
}

SIGNAL(UART0_TRANSMIT_INTERRUPT)
/*************************************************************************
Function: UART Data Register Empty interrupt
Purpose:  called when the UART is ready to transmit the next byte
*************************************************************************/
{
    unsigned char tmptail;


    if ( UART_TxHead != UART_TxTail) {
        /* calculate and store new buffer index */
        tmptail = (UART_TxTail + 1) & UART_TX_BUFFER_MASK;
        UART_TxTail = tmptail;
        /* get one byte from buffer and write it to UART */
        UART0_DATA = UART_TxBuf[tmptail];  /* start transmission */
    }else{
        /* tx buffer empty, disable UDRE interrupt */
        UART0_CONTROL &= ~_BV(UART0_UDRIE);
    }
}

/*************************************************************************
Function: uart_init()
Purpose:  initialize UART and set baudrate
Input:    baudrate using macro UART_BAUD_SELECT()
Returns:  none
*************************************************************************/
void uart_init(unsigned int baudrate)
{
    UART_TxHead = 0;
    UART_TxTail = 0;
    UART_RxHead = 0;
    UART_RxTail = 0;

#if defined( AT90_UART )
    /* set baud rate */
    UBRR = (unsigned char)baudrate;

    /* enable UART receiver and transmmitter and receive complete interrupt */
    UART0_CONTROL = _BV(RXCIE)|_BV(RXEN)|_BV(TXEN);

#elif defined (ATMEGA_USART)
    /* Set baud rate */
    if ( baudrate & 0x8000 )
    {
         UART0_STATUS = (1<<U2X);  //Enable 2x speed
         baudrate &= ~0x8000;
    }
    UBRRH = (unsigned char)(baudrate>>8);
    UBRRL = (unsigned char) baudrate;

    /* Enable USART receiver and transmitter and receive complete interrupt */
    UART0_CONTROL = _BV(RXCIE)|(1<<RXEN)|(1<<TXEN);

    /* Set frame format: asynchronous, 8data, no parity, 1stop bit */
    #ifdef URSEL
    UCSRC = (1<<URSEL)|(3<<UCSZ0);
    #else
    UCSRC = (3<<UCSZ0);
    #endif

#elif defined (ATMEGA_USART0 )
    /* Set baud rate */
    if ( baudrate & 0x8000 )
    {
        UART0_STATUS = (1<<U2X0);  //Enable 2x speed
        baudrate &= ~0x8000;
    }
    UBRR0H = (unsigned char)(baudrate>>8);
    UBRR0L = (unsigned char) baudrate;

    /* Enable USART receiver and transmitter and receive complete interrupt */
    UART0_CONTROL = _BV(RXCIE0)|(1<<RXEN0)|(1<<TXEN0);

    /* Set frame format: asynchronous, 8data, no parity, 1stop bit */
    #ifdef URSEL0
    UCSR0C = (1<<URSEL0)|(3<<UCSZ00);
```

```
    #else
    UCSR0C = (3<<UCSZ00);
    #endif

#elif defined ( ATMEGA_UART )
    /* set baud rate */
    if ( baudrate & 0x8000 )
    {
        UART0_STATUS = (1<<U2X);  //Enable 2x speed
        baudrate &= ~0x8000;
    }
    UBRRHI = (unsigned char)(baudrate>>8);
    UBRR   = (unsigned char) baudrate;

    /* Enable UART receiver and transmitter and receive complete interrupt */
    UART0_CONTROL = _BV(RXCIE)|(1<<RXEN)|(1<<TXEN);

#endif

}/* uart_init */


/*************************************************************************
Function: uart_getc()
Purpose:  return byte from ringbuffer
Returns:  lower byte:  received byte from ringbuffer
          higher byte: last receive error
*************************************************************************/
unsigned int uart_getc(void)
{
    unsigned char tmptail;
    unsigned char data;


    if ( UART_RxHead == UART_RxTail ) {
        return UART_NO_DATA;   /* no data available */
    }

    /* calculate /store buffer index */
    tmptail = (UART_RxTail + 1) & UART_RX_BUFFER_MASK;
    UART_RxTail = tmptail;

    /* get data from receive buffer */
    data = UART_RxBuf[tmptail];

    return (UART_LastRxError << 8) + data;
}/* uart_getc */

/*************************************************************************
Function: uart_putc()
Purpose:  write byte to ringbuffer for transmitting via UART
Input:    byte to be transmitted
Returns:  none
*************************************************************************/
void uart_putc(unsigned char data)
{
    unsigned char tmphead;


    tmphead  = (UART_TxHead + 1) & UART_TX_BUFFER_MASK;

    while ( tmphead == UART_TxTail ){
        ;/* wait for free space in buffer */
    }

    UART_TxBuf[tmphead] = data;
    UART_TxHead = tmphead;

    /* enable UDRE interrupt */
    UART0_CONTROL    |= _BV(UART0_UDRIE);

}/* uart_putc */

/*************************************************************************
Function: uart_puts()
Purpose:  transmit string to UART
Input:    string to be transmitted
Returns:  none
*************************************************************************/
void uart_puts(const char *s )
{
    while (*s)
      uart_putc(*s++);

}/* uart_puts */

/*************************************************************************
Function: uart_puts_p()
Purpose:  transmit string from program memory to UART
Input:    program memory string to be transmitted
Returns:  none
*************************************************************************/
```

```
void uart_puts_p(const char *progmem_s )
{
    register char c;

    while ( (c = pgm_read_byte(progmem_s++)) )
      uart_putc(c);

}/* uart_puts_p */


/*
 * these functions are only for ATmegas with two USART
 */
#if defined( ATMEGA_USART1 )

SIGNAL(UART1_RECEIVE_INTERRUPT)
/*************************************************************************
Function: UART1 Receive Complete interrupt
Purpose:  called when the UART1 has received a character
*************************************************************************/
{
    unsigned char tmphead;
    unsigned char data;
    unsigned char usr;
    unsigned char lastRxError;


    /* read UART status register and UART data register */
    usr  = UART1_STATUS;
    data = UART1_DATA;

    /* */
    lastRxError = (usr & (_BV(FE1)|_BV(DOR1)) );

    /* calculate buffer index */
    tmphead = ( UART1_RxHead + 1) & UART_RX_BUFFER_MASK;

    if ( tmphead == UART1_RxTail ) {
        /* error: receive buffer overflow */
        lastRxError = UART_BUFFER_OVERFLOW >> 8;
    }else{
        /* store new index */
        UART1_RxHead = tmphead;
        /* store received data in buffer */
        UART1_RxBuf[tmphead] = data;
    }
    UART1_LastRxError = lastRxError;
}

SIGNAL(UART1_TRANSMIT_INTERRUPT)
/*************************************************************************
Function: UART1 Data Register Empty interrupt
Purpose:  called when the UART1 is ready to transmit the next byte
*************************************************************************/
{
    unsigned char tmptail;


    if ( UART1_TxHead != UART1_TxTail) {
        /* calculate and store new buffer index */
        tmptail = (UART1_TxTail + 1) & UART_TX_BUFFER_MASK;
        UART1_TxTail = tmptail;
        /* get one byte from buffer and write it to UART */
        UART1_DATA = UART1_TxBuf[tmptail];  /* start transmission */
    }else{
        /* tx buffer empty, disable UDRE interrupt */
        UART1_CONTROL &= ~_BV(UART1_UDRIE);
    }
}

/*************************************************************************
Function: uart1_init()
Purpose:  initialize UART1 and set baudrate
Input:    baudrate using macro UART_BAUD_SELECT()
Returns:  none
*************************************************************************/
void uart1_init(unsigned int baudrate)
{
    UART1_TxHead = 0;
    UART1_TxTail = 0;
    UART1_RxHead = 0;
    UART1_RxTail = 0;

    /* Set baud rate */
    if ( baudrate & 0x8000 )
    {
        UART1_STATUS = (1<<U2X1);  //Enable 2x speed
      baudrate &= ~0x8000;
    }
    UBRR1H = (unsigned char)(baudrate>>8);
    UBRR1L = (unsigned char) baudrate;
```

```
    /* Enable USART receiver and transmitter and receive complete interrupt
    UART1_CONTROL = _BV(RXCIE1)|(1<<RXEN1)|(1<<TXEN1);

    /* Set frame format: asynchronous, 8data, no parity, 1stop bit */
    #ifdef URSEL1
    UCSR1C = (1<<URSEL1)|(3<<UCSZ10);
    #else
    UCSR1C = (3<<UCSZ10);
    #endif
}/* uart_init */

/*************************************************************************
Function: uart1_getc()
Purpose:  return byte from ringbuffer
Returns:  lower byte:  received byte from ringbuffer
          higher byte: last receive error
*************************************************************************/
unsigned int uart1_getc(void)
{
    unsigned char tmptail;
    unsigned char data;

    if ( UART1_RxHead == UART1_RxTail )
    {
        return UART_NO_DATA;   /* no data available */
    }

    /* calculate /store buffer index */
    tmptail = (UART1_RxTail + 1) & UART_RX_BUFFER_MASK;
    UART1_RxTail = tmptail;

    /* get data from receive buffer */
    data = UART1_RxBuf[tmptail];
    return (UART1_LastRxError << 8) + data;

}/* uart1_getc */

/*************************************************************************
Function: uart1_putc()
Purpose:  write byte to ringbuffer for transmitting via UART
Input:    byte to be transmitted
Returns:  none
*************************************************************************/
void uart1_putc(unsigned char data)
{
    unsigned char tmphead;

    tmphead  = (UART1_TxHead + 1) & UART_TX_BUFFER_MASK;

    while ( tmphead == UART1_TxTail ){
        ;/* wait for free space in buffer */
    }

    UART1_TxBuf[tmphead] = data;
    UART1_TxHead = tmphead;

    /* enable UDRE interrupt */
    UART1_CONTROL    |= _BV(UART1_UDRIE);

}/* uart1_putc */

/*************************************************************************
Function: uart1_puts()
Purpose:  transmit string to UART1
Input:    string to be transmitted
Returns:  none
*************************************************************************/
void uart1_puts(const char *s )
{
    while (*s)
      uart1_putc(*s++);

}/* uart1_puts */


/*************************************************************************
Function: uart1_puts_p()
Purpose:  transmit string from program memory to UART1
Input:    program memory string to be transmitted
Returns:  none
*************************************************************************/
void uart1_puts_p(const char *progmem_s )
{
    register char c;

    while ( (c = pgm_read_byte(progmem_s++)) )
      uart1_putc(c);

}/* uart1_puts_p */


#endif
```

Uart.h subprogram

```
#define UART_H

#define RBUFLEN 255 //Pufferlänge für seriellen Empfang

extern volatile unsigned char rcnt, rpos;
extern volatile unsigned char busy;
#define F_CPU 8000000UL
#define UART_BAUD_RATE      19200
#define UART_BAUD_SELECT    (F_CPU/(UART_BAUD_RATE*16l)-1)

#if defined (__AVR_ATmega128__) || defined (__AVR_ATmega64__)
        || defined (__AVR_AT90CAN128__)
  #define TX0_PIN   1
  #define TX0_PORT  PORTE
  #define TX0_DDR   DDRE

  #define SIG_UART0_RECEIVE             SIG_UART0_RECV
  #define SIG_UART0_TRANSMIT            SIG_UART0_TRANS

  #define UART0_RECEIVE_REGISTER        UDR0
  #define UART0_TRANSMIT_REGISTER       UDR0
  #define UART1_RECEIVE_REGISTER        UDR1
  #define UART1_TRANSMIT_REGISTER       UDR1

  #define ENABLE_UART0_TRANSMIT_INT     sbi(UCSR0B,TXEN)
  #define DISABLE_UART0_TRANSMIT_INT    cbi(UCSR0B,TXEN);
  #define ENABLE_UART0_RECEIVE_INT      sbi(UCSR0B,RXCIE)
  #define DISABLE_UART0_RECEIVE_INT     cbi(UCSR0B,RXCIE)
  #define ENABLE_UART1_TRANSMIT_INT     sbi(UCSR1B,TXEN)
  #define DISABLE_UART1_TRANSMIT_INT    cbi(UCSR1B,TXEN);
  #define ENABLE_UART1_RECEIVE_INT      sbi(UCSR1B,RXCIE)
  #define DISABLE_UART1_RECEIVE_INT     cbi(UCSR1B,RXCIE)

  #define ENABLE_UART0_RECEIVER     sbi(UCSR0B,RXEN0)
  #define DISABLE_UART0_RECEIVER    cbi(UCSR0B,RXEN0)
  #define ENABLE_UART1_RECEIVER     sbi(UCSR1B,RXEN1)
  #define DISABLE_UART1_RECEIVER    cbi(UCSR1B,RXEN1)

  #define UART0_BAUD_REGISTER_HIGH      UBRR0H
  #define UART0_BAUD_REGISTER_LOW       UBRR0L
  #define UART1_BAUD_REGISTER_HIGH      UBRR1H
  #define UART1_BAUD_REGISTER_LOW       UBRR1L


  #define UART0_CONFIGURE1              UCSR0B= (1<<RXCIE) | (1<<TXCIE) |
                                                (1<<RXEN) | (1<<TXEN)

  #define UART0_CONFIGURE2             UCSR0C= (1<<UCSZ1) | (1<<UCSZ0)
  //8 Bit,1 Stop, no parity

#else
//#  error "processor type not defined in serial.h"
#endif


#if (__GNUC__ * 100 + __GNUC_MINOR__) < 304
#error "This library requires AVR-GCC 3.4 or later, update to newer AVR-GCC c
#endif

/*
** constants and macros
*/

/** @brief  UART Baudrate Expression
 *  @param  xtalcpu  system clock in Mhz, e.g. 4000000L for 4Mhz
 *  @param  baudrate baudrate in bps, e.g. 1200, 2400, 9600
 */
#define UART_BAUD_SELECT(baudRate,xtalCpu) ((xtalCpu)/((baudRate*16l))-1)

/** @brief  UART Baudrate Expression for ATmega double speed mode
 *  @param  xtalcpu  system clock in Mhz, e.g. 4000000L for 4Mhz
 *  @param  baudrate baudrate in bps, e.g. 1200, 2400, 9600
 */
#define UART_BAUD_SELECT_DOUBLE_SPEED(baudRate,xtalCpu)
        (((xtalCpu)/((baudRate)*8l)-1)|0x8000)

/** Size of the circular receive buffer, must be power of 2 */
#ifndef UART_RX_BUFFER_SIZE
#define UART_RX_BUFFER_SIZE 32
#endif
/** Size of the circular transmit buffer, must be power of 2 */
#ifndef UART_TX_BUFFER_SIZE
#define UART_TX_BUFFER_SIZE 32
#endif

/* test if the size of the circular buffers fits into SRAM */
#if ( (UART_RX_BUFFER_SIZE+UART_TX_BUFFER_SIZE) >= (RAMEND-0x60 ) )
#error "size of UART_RX_BUFFER_SIZE + UART_TX_BUFFER_SIZE larger than size of
#endif

/*
** high byte error return code of uart_getc()
*/
```

```
#define UART_FRAME_ERROR      0x0800              /* Framing Error by UART
#define UART_OVERRUN_ERROR    0x0400              /* Overrun condition by UAR
#define UART_BUFFER_OVERFLOW  0x0200              /* receive ringbuffer overf
#define UART_NO_DATA          0x0100              /* no receive data availabl

/*
** function prototypes
*/

/**
   @brief   Initialize UART and set baudrate
   @param   baudrate Specify baudrate using macro UART_BAUD_SELECT()
   @return  none
*/
extern void uart_init(unsigned int baudrate);

/**
 *  @brief   Get received byte from ringbuffer
 *
 * Returns in the lower byte the received character and in the
 * higher byte the last receive error.
 * UART_NO_DATA is returned when no data is available.
 *
 *  @param   void
 *  @return  lower byte:  received byte from ringbuffer
 *  @return  higher byte: last receive status
 *           - \b 0 successfully received data from UART
 *           - \b UART_NO_DATA
 *             <br>no receive data available
 *           - \b UART_BUFFER_OVERFLOW
 *             <br>Receive ringbuffer overflow.
 *             We are not reading the receive buffer fast enough,
 *             one or more received character have been dropped
 *           - \b UART_OVERRUN_ERROR
 *             <br>Overrun condition by UART.
 *             A character already present in the UART UDR register was
 *             not read by the interrupt handler before the next character ar
 *             one or more received characters have been dropped.
 *           - \b UART_FRAME_ERROR
 *             <br>Framing Error by UART
 */
extern unsigned int uart_getc(void);

/**
 *  @brief   Put byte to ringbuffer for transmitting via UART
 *  @param   data byte to be transmitted
 *  @return  none
 */
extern void uart_putc(unsigned char data);

/**
 *  @brief   Put string to ringbuffer for transmitting via UART
 *
 *  The string is buffered by the uart library in a circular buffer
 *  and one character at a time is transmitted to the UART using interrupts.
 *  Blocks if it can not write the whole string into the circular buffer.
 *
 *  @param   s string to be transmitted
 *  @return  none
 */
extern void uart_puts(const char *s );

/**
 * @brief    Put string from program memory to ringbuffer for transmitting vi
 *
 * The string is buffered by the uart library in a circular buffer
 * and one character at a time is transmitted to the UART using interrupts.
 * Blocks if it can not write the whole string into the circular buffer.
 *
 * @param    s program memory string to be transmitted
 * @return   none
 * @see      uart_puts_P
 */
extern void uart_puts_p(const char *s );

/**
 * @brief    Macro to automatically put a string constant into program memory
 */
#define uart_puts_P(__s)       uart_puts_p(PSTR(__s))

/** @brief  Initialize USART1 (only available on selected ATmegas) @see uart_
extern void uart1_init(unsigned int baudrate);
/** @brief  Get received byte of USART1 from ringbuffer. (only available on s
extern unsigned int uart1_getc(void);
/** @brief  Put byte to ringbuffer for transmitting via USART1 (only availabl
extern void uart1_putc(unsigned char data);
/** @brief  Put string to ringbuffer for transmitting via USART1 (only availa
extern void uart1_puts(const char *s );
/** @brief  Put string from program memory to ringbuffer for transmitting via
extern void uart1_puts_p(const char *s );
/** @brief  Macro to automatically put a string constant into program memory
#define uart1_puts_P(__s)       uart1_puts_p(PSTR(__s))

/**@}*/

#endif // UART_H
```

ADC.c subprogram

```
#include <avr/io.h>
#include <util/delay.h>
#include "ADC.h"
#include "uart.h"

unsigned int adc_data;
unsigned int adch_low,adch_high,adcl;
unsigned int adch_value,adcl_value,adc_value,Mode=0;
unsigned int adc[10];
int Pressure=0;

void init_ADC()
{
    DDRB = 0xFF;  //pull up
    PORTB = 0x00; //switch ofF all led's

    ADMUX= (1<<ADLAR);
    ADCSRA = 0x84; // Enable ADC and Setting Division Factor 0100
    ADMUX |=0xC0; //internal 2.56 V voltage reference REFS1=1 REFS0=1,
    //channel selection ADC0
    ADCSRA |= 0X80; //ENABLE ADC
}

void read_ADC()
{
    ADCSRA|= 0x40; //starting the conversion
    while(ADCSRA & 0x40); // wait for the end of the conversion
    //sleep_ms(1);
    adcl_value=ADCL;
    //adcl_value=0xFF;
    adcl= (adcl_value & 0xF0);
    adcl= (adcl >> 6);
    adcl= adcl+0x30;
    //-------------
    adch_value=ADCH;
    //adch_value=0xC0;
    adch_low=(adch_value & 0x0F);
    adch_high= (adch_value & 0xF0);
    adch_high= (adch_high >> 4);
    if (adch_low<10) adch_low=adch_low+0x30;
    else adch_low=adch_low+0x37;
    if (adch_high<10) adch_high=adch_high+0x30;
    else adch_high=adch_high+0x37;
    PORTB = adch_value;
    display_ADC();
    send_data();
}

void send_data(void)
{
    uart_putc(adc[0]+0x30);
    uart_putc(adc[1]+0x30);
    uart_putc(adc[2]+0x30);
    uart_putc(0x20);
}


void display_ADC(void)
{
  adc[0]=0;
  adcl_value=(adcl_value >> 6);
  adc_value=adcl_value*256+adch_value;
  //adc_value=972;
  //---------------------------------------- display Value
  //adc_value=1024;  // test Equ
  Pressure=round(((adc_value*5/1023)*19.86)-50.917);
  if (Pressure<0)
    {
        adc[0]=1;
        Pressure=Pressure*(-1);
        lcd_gotoxy(13,0);
        lcd_putc(0x2D);
    }
  adc[1]=Pressure/10;
  adc[2]=Pressure%10;
  lcd_gotoxy(14,0);
  lcd_putc(adc[1]+0x30);
  lcd_gotoxy(15,0);
  lcd_putc(adc[2]+0x30);

}
```

ADC.h subprogram

```
void init_ADC(void);
void read_ADC(void);
void display_ADC(void);
void send_data(void);
```

Multiplexer.c program

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>
#include "uart.h"
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <avr/pgmspace.h>
//------------------------
#include "Multiplexer.h"

unsigned int RXData=0,RXDataTemp=0,loop=0,loopTemp=0, sum_data=0,
firstnumber=0,secondnumber=0;
unsigned int mux_Select=0,mux_SelectTemp=0;
unsigned int str_data[10];

void Serial_Config(void)
{
    uart_init(UART_BAUD_SELECT(38400,F_CPU));
    //ENABLE_UART0_RECEIVER;
}
void Test(void)
{
 firstnumber=2;
 secondnumber=5;
 mux_Step();

}

void RXData_Labview(void)
{
//ENABLE_UART0_RECEIVER;
labelA:
    RXData = uart_getc();
    if (RXData & UART_NO_DATA) goto end_RXData;
    else if (RXData!='$') goto end_RXData;
    else
    {
labelB:
        _delay_us(1000);
        RXData = uart_getc();
        if (RXData & UART_NO_DATA) goto labelB;
        else if (RXData=='0' || RXData=='1' || RXData=='2' || RXData== '3'
                 || RXData== '4'|| RXData== '5' || RXData== '6' ||
                 RXData== '7'
                 || RXData== '8'|| RXData== '9')
            {
```

```
            //lcd_clrscr();
            sum_data++;
            str_data[sum_data]=RXData;
            if (sum_data>1)
            {
               firstnumber=str_data[1]-48;
               secondnumber=str_data[2]-48;
               mux_Select=firstnumber*10+secondnumber;
               //mux_Step();
               Mux();
               goto end_RXData;
            }
            goto labelB;
           }
     }
end_RXData:
    sum_data=0;
}

void detect_Home(void)
{
wait_Home:
if ( DH_Val() != 0)
 {
    _delay_us(500000);
    _delay_us(500000);
    _delay_us(500000);
    _delay_us(500000);
    _delay_us(500000);
    _delay_us(500000);
    if ( DH_Val() != 0)
    {
        lcd_gotoxy(12,1);
        lcd_puts("    ");
        lcd_gotoxy(12,1);
        lcd_puts("HOME");
        uart_puts("H");
        Home_Off();
    }
    else goto wait_Home;
 }
else goto wait_Home;
}

void Mux(void)
{
```

```
  loop=0;
  if (mux_Select==0)
  {
    mux_Home();
  }
  else if (mux_SelectTemp<mux_Select)
  {
    for (loop=mux_SelectTemp; loop < mux_Select; loop++)
    {
      mux_Step();
    }
    mux_SelectTemp=mux_Select;
  }
  else if (mux_SelectTemp>mux_Select)
  {
    mux_Home();
    mux_SelectTemp=0;
    for (loop=mux_SelectTemp; loop < mux_Select; loop++)
    {
      mux_Step();
    }
    mux_SelectTemp=mux_Select;
  }
}
void mux_Home(void)
{
    Home_On();
    detect_Home();
}

void mux_Step(void)
{
    loopTemp=loop+1;
    lcd_gotoxy(12,1);
    lcd_puts("    ");
    lcd_gotoxy(14,1);
    //lcd_putc(firstnumber+0x30);
    lcd_putc((loopTemp/10)+0x30);
    lcd_gotoxy(15,1);
    //lcd_putc(secondnumber+0x30);
    lcd_putc((loopTemp%10)+0x30);
    Step_On();
    _delay_us(100000);
    Step_Off();
    _delay_us(100000);
}
```

Multiplexer.h program

```
void Serial_Config(void);
void RXData_Labview(void);
void Mux(void);
void detect_Home(void);
void send_data(void);
void mux_Home(void);
void mux_Step(void);
void Test(void);


#define Home      (1 << PA6)
#define Home_On()  PORTA |=  Home
#define Home_Off() PORTA &= ~Home

#define Step      (1 << PA7)
#define Step_On()  PORTA |=  Step
#define Step_Off() PORTA &=  ~Step

#define DH        (1 << PA1)
#define DH_On()   (PORTA |= DH)
#define DH_Val()  (PINA & DH)
```